

普通高等教育“十一五”国家级规划教材  
信息系统工程丛书

# 信息系统原理与工程 (第3版)

张维明 戴长华 封孝生 等编著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

## 内 容 简 介

本书主要讲述信息系统的概念、基本原理和模型方法,比较全面地介绍了信息系统的理论体系和分析、设计、开发方法,包括结构化方法、面向对象方法等工程技术。全书共8章,内容分别为:信息系统概述,信息系统的基础理论,信息系统的开发,信息系统的战略规划与可行性研究,结构化系统分析与设计,面向对象系统分析与设计,系统实施和信息系统项目管理。本书内容新颖,理论体系完整,可操作性强。

本书可作为高等院校信息系统工程、计算机信息管理、管理工程等专业的本科生教材,也可作为管理人员和计算机应用人员的参考书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

图书在版编目(CIP)数据

信息系统原理与工程/张维明等编著. —3版. —北京:电子工业出版社,2009.1

(信息工程丛书)

ISBN 978-7-121-07814-9

I. 信… II. 张 … III. 信息系统—系统工程 IV. C931.6

中国版本图书馆 CIP 数据核字(2008)第 190286 号

策划编辑:秦 梅

责任编辑:韩玲玲

印 刷:

装 订:

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×980 1/16 印张:26.75 字数:538.4 千字

印 次:2009 年 1 月第 1 次印刷

印 数:3 000 册 定价:35.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010)88254888。

质量投诉请发邮件至 [zlt@phei.com.cn](mailto:zlt@phei.com.cn),盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线:(010)88258888。

# 前 言

信息系统是信息科学、管理科学、系统科学、计算机科学与通信技术相结合的综合性的、交叉性的、具有独特风格的边缘学科,同时它又是一门实践性很强的应用科学,在实践中产生,又在实践中不断发展并完善。它的主要任务是研究信息处理过程的内在规律,以及基于计算机等现代化手段的形式化表达和处理规律。经过人们不断地探索和实践,目前已形成了信息系统独具特色的理论和技术体系,其应用的触角已深入到社会生活的各个方面。以信息系统为中心的信息产业已成为当今信息化社会最活跃、最有生机、最有潜力的支柱产业之一。

信息系统工程是信息系统科学中的一个重要组成部分。信息系统工程是用系统工程的原理、方法来指导信息系统建设与管理的一门工程技术学科。它的基本概念、原理和方法,对实际分析、设计和开发一个信息系统,从理论、手段、方法、技术等多方面提供了一套完整、科学、适用的研究与开发体系,具有十分重要的应用价值,对实际建设信息系统有着重要的理论指导意义。本书主要向读者介绍了信息系统工程的基本概念、原理、方法和技术,以及如何运用系统工程思想和现代计算机技术完成信息系统的分析、设计和实施。书中从理论和实践两个方面探讨了如何运用这些原理、方法和技术建设信息系统。

本书内容由四部分组成:第一部分(第1、2章)为概念部分,主要介绍了系统、信息、信息系统和信息系统工程的基本概念,以及信息系统的数学基础;第二部分(第3章)对信息系统的开发方法概括地介绍;第三部分(第4、5、6、7章)详细介绍了信息系统开发过程的几个主要阶段和各个阶段中的技术方法,它是信息系统工程的主要内容和核心;第四部分(第8章)介绍了信息系统建设中的项目管理问题。

本书可作为高等院校信息系统工程、计算机信息管理、管理工程等专业的本科生教材,也可作为信息系统规划、开发、管理人员及计算机软件开发人员的参考书。

参加本书编写的有张维明、戴长华、封孝生、肖卫东、钱猛、陈卫东。由于编者水平有限,书中难免存在一些缺点和欠妥之处,恳切希望广大读者批评指正。

作 者

# 目 录

第 1 章 信息系统概述 .....	1
1.1 系统 .....	1
1.1.1 系统的概念 .....	2
1.1.2 系统的特性 .....	4
1.1.3 系统的一般模型 .....	5
1.1.4 系统层次与系统分类 .....	8
1.1.5 系统学基本原理 .....	11
1.1.6 系统性能和标准 .....	16
1.2 信息 .....	17
1.2.1 信息的概念 .....	17
1.2.2 信息的性质 .....	19
1.2.3 信息运动模型 .....	21
1.2.4 信息的度量 .....	23
1.2.5 信息的分类 .....	30
1.2.6 信息的特点与价值 .....	30
1.2.7 信息与管理 .....	32
1.3 信息系统 .....	34
1.3.1 信息系统的概念 .....	34
1.3.2 信息系统的发展 .....	36
1.3.3 信息系统的基本功能 .....	38
1.3.4 信息系统的结构 .....	39
1.3.5 信息系统的价值 .....	42
1.3.6 信息系统的评价 .....	43
1.4 信息系统工程 .....	45
1.4.1 基本概念 .....	45
1.4.2 信息系统工程的研究方法 .....	46
1.4.3 信息系统工程的研究范围 .....	47
习题 .....	48



第 2 章 信息系统的基础理论 .....	49
2.1 集合论基础 .....	49
2.1.1 集合及其表示 .....	49
2.1.2 集合的运算与笛卡儿乘积 .....	50
2.1.3 关系 .....	51
2.1.4 关系的闭包 .....	52
2.1.5 函数映射 .....	52
2.1.6 集合的划分 .....	53
2.2 信息系统的数学模型 .....	53
2.3 信息系统的性质 .....	55
2.3.1 对象的信息 .....	55
2.3.2 最大的信息系统 .....	56
2.3.3 可选的信息系统 .....	57
2.3.4 信息系统的划分 .....	57
2.3.5 最简信息系统 .....	62
2.3.6 理想信息系统 .....	68
2.3.7 信息系统中的决策规则 .....	69
2.3.8 不确定性信息系统 .....	70
2.4 信息系统的连接 .....	73
2.4.1 良好的连接系统 .....	73
2.4.2 连接系统的分类 .....	74
2.4.3 连接系统的基本性质 .....	76
习题 .....	76
第 3 章 信息系统的开发 .....	77
3.1 信息系统生命周期 .....	77
3.2 常用信息系统开发方法 .....	80
3.2.1 结构化开发方法 .....	80
3.2.2 面向对象开发方法 .....	81
3.2.3 原型化开发方法 .....	81
3.3 信息系统开发的过程模型 .....	86
3.3.1 瀑布模型 .....	87
3.3.2 原型模型 .....	88
3.3.3 RAD 模型 .....	89

3.3.4	增量模型	90
3.3.5	螺旋模型	91
3.3.6	构件组装模型	92
3.3.7	组合模型	92
3.3.8	形式化方法模型	93
3.4	信息系统开发方法学	93
3.4.1	系统开发认知体系	94
3.4.2	系统开发方法学	94
3.4.3	系统开发策略与资源规划	95
3.4.4	信息系统开发方法的规范化研究	95
	习题	95
第4章	信息系统的战略规划与可行性研究	97
4.1	信息系统战略规划的概念、目标与组织	97
4.1.1	信息系统战略规划的概念与层次	97
4.1.2	信息系统战略规划的目标、作用、内容与组织	101
4.2	信息系统战略规划的步骤	103
4.2.1	诺兰的阶段模型	103
4.2.2	信息系统战略规划的三阶段模型	106
4.2.3	信息系统战略规划的步骤	106
4.3	信息系统战略规划的常用方法	107
4.4	信息工程与战略数据规划	117
4.4.1	信息工程的基本原理	117
4.4.2	信息工程方法论	119
4.4.3	战略数据规划的目标与步骤	120
4.5	可行性研究	123
4.5.1	明确要求	123
4.5.2	环境调查	124
4.5.3	提出方案	125
4.5.4	可行性分析	126
4.6	基于体系结构的信息系统顶层设计	128
	习题	130
第5章	结构化系统分析与设计	131
5.1	结构化方法的基本思想	131

5.2	结构化分析概述 .....	132
5.2.1	系统分析的任务 .....	132
5.2.2	结构化分析的方法 .....	134
5.2.3	结构化分析的工具 .....	135
5.2.4	系统要求的确定 .....	136
5.3	数据流分析技术 .....	139
5.3.1	数据流分析 .....	139
5.3.2	数据流图 .....	140
5.3.3	数据流图的建立 .....	142
5.3.4	数据字典 .....	147
5.4	IDEF0 分析技术 .....	152
5.4.1	IDEF0 的特点 .....	153
5.4.2	功能模型的表示 .....	156
5.4.3	IDEF0 建模过程 .....	160
5.4.4	IDEF0 分析方法与数据流分析方法的比较 .....	163
5.5	逻辑分析工具 .....	163
5.5.1	决策树 .....	165
5.5.2	决策表 .....	166
5.5.3	结构式语言 .....	167
5.5.4	三种逻辑分析工具的比较 .....	169
5.6	系统设计概述 .....	169
5.6.1	系统设计的任务 .....	170
5.6.2	系统设计的目标 .....	170
5.6.3	计算机处理与手工处理 .....	173
5.7	结构化设计原理 .....	174
5.7.1	结构化设计方法 .....	174
5.7.2	结构化设计原理 .....	175
5.8	模块化设计 .....	176
5.8.1	模块 .....	177
5.8.2	模块的耦合 .....	180
5.8.3	模块的聚合 .....	182
5.8.4	若干其他设计原则及有益的建议 .....	185
5.9	面向数据流的设计 .....	189

5.9.1	结构图 .....	189
5.9.2	设计过程 .....	190
5.9.3	设计优化 .....	199
习题 .....		200
<b>第 6 章 面向对象系统分析与设计</b> .....		203
6.1	面向对象的基本概念 .....	203
6.1.1	对象 .....	203
6.1.2	消息 .....	204
6.1.3	类 .....	204
6.1.4	继承 .....	205
6.1.5	封装 .....	206
6.1.6	多态 .....	207
6.2	面向对象的方法论 .....	208
6.2.1	面向对象的发展过程 .....	208
6.2.2	分析方法的改进 .....	210
6.2.3	从认识论看面向对象 .....	212
6.2.4	面向对象的几种方法 .....	217
6.3	面向对象的分析 .....	221
6.3.1	面向对象的分析原则 .....	222
6.3.2	面向对象分析的模型和过程 .....	224
6.3.3	标识发现对象 .....	226
6.3.4	定义属性 .....	233
6.3.5	定义方法 .....	235
6.3.6	标识结构和连接 .....	238
6.3.7	定义主题 .....	248
6.4	面向对象的设计 .....	255
6.4.1	问题域部分的设计 .....	256
6.4.2	人机交互部分的设计 .....	263
6.4.3	任务管理部分的设计 .....	268
6.4.4	数据管理部分的设计 .....	271
6.4.5	面向对象设计的评价标准 .....	278
6.5	统一建模语言(UML) .....	282
6.5.1	UML 的发展过程 .....	283

6.5.2	UML 的特点 .....	284
6.5.3	UML 的模型元素和构成 .....	286
6.5.4	UML 建模的一般方法 .....	287
6.5.5	UML 一般建模过程 .....	289
6.6	用例驱动的结构建模和行为建模 .....	294
6.6.1	基于过程的 UML 建模方法论 .....	294
6.6.2	用例建模 .....	303
6.6.3	结构建模 .....	305
6.6.4	行为建模 .....	312
6.7	UML 建模工具简介 .....	317
6.7.1	概况 .....	317
6.7.2	几种典型的 UML 建模工具 .....	318
	习题 .....	320
第 7 章	系统实施 .....	323
7.1	概述 .....	323
7.2	开发平台 .....	325
7.2.1	J2EE 平台 .....	326
7.2.2	.NET 平台 .....	331
7.2.3	J2EE 和 .NET 平台的异同 .....	337
7.3	系统编程 .....	340
7.3.1	编程概述 .....	340
7.3.2	编程语言 .....	341
7.3.3	编程风格 .....	342
7.4	系统测试 .....	344
7.4.1	测试概述 .....	344
7.4.2	系统测试方法 .....	347
7.4.3	系统测试过程 .....	352
7.4.4	系统测试工具 .....	360
7.5	运行与维护 .....	362
7.5.1	系统运行 .....	362
7.5.2	系统维护 .....	364
7.5.3	维护过程 .....	365
7.5.4	维护的特点 .....	368

7.5.5	可维护性 .....	370
7.5.6	软件重用与系统维护 .....	374
7.5.7	信息系统的质量维护 .....	376
习题 .....		377
第 8 章	信息系统项目管理 .....	379
8.1	项目管理概述 .....	379
8.1.1	项目管理的概念 .....	379
8.1.2	项目管理的基本内容和特点 .....	380
8.1.3	项目管理知识体系 .....	382
8.2	信息系统的项目管理 .....	384
8.2.1	概述 .....	385
8.2.2	基本内容与步骤 .....	386
8.3	信息系统项目时间管理 .....	392
8.3.1	时间管理流程 .....	392
8.3.2	工程进度管理工具和技术 .....	394
8.4	信息系统项目人力资源管理 .....	397
8.4.1	项目管理的组织机构 .....	397
8.4.2	项目角色及其职责 .....	399
8.4.3	管理中的协调工作 .....	403
8.5	信息系统项目质量管理 .....	404
8.5.1	信息系统质量管理概述 .....	404
8.5.2	信息系统质量控制的组织职能 .....	406
8.5.3	项目开发的质量控制 .....	407
8.6	信息系统开发的文档管理 .....	408
8.6.1	信息系统的质量维护文档的内容与分类 .....	409
8.6.2	文档的规范化管理 .....	411
习题 .....		413
参考文献 .....		415

# 第 1 章 信息系统概述

现代科学技术的飞速发展，使人类认识和理解客观世界的能力和手段发生了质的变化。信息技术的出现大大改变了人类生活和工作的方式。这一切都得益于 20 世纪人们对信息、信息系统的认识和研究。信息系统是关于信息的系统，它是在客观世界的真实系统中的“神经”系统，同时也是信息系统工程研究的对象。本章着重介绍系统、信息和信息系统的基本概念和基本原理。

## 1.1 系统

系统的概念来源于人类长期的社会实践。由于受到科学技术发展初始阶段局限性的影响，所以这个概念一直没有受到重视。直到 20 世纪 40 年代，人们才开始逐渐地认识和应用它。当时，人们在一些科学学科的研究中，尤其是在生物学、心理学和社会科学中，发现系统的一些固有性质与个别系统的特殊性无关，也就是说，若以传统的科学分类为基础研究，则无法发现和搞清系统的主要性质。奥地利生物学家路德维希·冯·贝塔朗菲（L. V. Bertalanffy）在 20 世纪 30~40 年代的一系列研究中提出了一般系统概念和一般系统理论，系统才逐渐被人们认为是一种综合性的学科。

一般系统理论研究系统与系统、系统与环境之间的普遍联系，研究各类系统运动带有规律性的思想、理论、方法和工具。它的任务是确立适用于各种系统的一般原则，不能把它局限在“技术”范围内，也不能把它当作一种数学理论来看待，因为有许多系统问题不能用现代数学概念求出解答，而要从系统观点来认识和分析客观事物。一般系统理论的研究领域十分广阔，几乎包括一切与系统有关的学科和理论，如管理科学、运筹学、信息论、控制论、科学学、行为科学、经济学等。它给各门学科带来新的动力和新的研究方法，同时也吸收其他学科的研究成果。它沟通了自然科学和社会科学、技术科学和人文科学之间的联系，促进了现代科学技术发展的整体化趋势。一般系统理论在发展过程中与系统工程有密切关系，它们相互促进、相互渗透，为人类走向系统时代奠定了理论基础。在逻辑上，系统工程是一般系统理论的实际应用，但在历史上系统工程又是一般系统理论的科学基础之一。

随着科学技术的发展，现代数学方法和计算机技术为系统理论提供了定量方法和强有力的计算工具，这就使一般系统理论与其他各种理论和系统分析方法紧密结合而逐渐发展成为系统科学。

目前，对于系统科学的一致认识是：系统科学是从系统的角度去考察和研究整个客观世界，为人类大规模认识和改造世界提供科学理论和方法的一门科学。系统科学可分为三个层次。

第一个层次是基础科学，即系统学。它主要研究系统的普遍属性、运动规律，系统间复杂关系的形成法则，系统结构和功能的关系，系统有序结构的形成规律及仿真的基本原理等。

第二个层次是技术科学。它主要研究系统共性问题的技术理论，如最优化理论。目前系统的技术科学主要是运筹学。

第三个层次是工程技术。系统的工程技术叫系统工程。它是直接改造客观世界的学问，是大系统的组织管理技术。由于系统的性质不同，故相应的系统工程的内容也不完全相同，如信息系统工程、军事系统工程、教育系统工程、经济系统工程等。

### 1.1.1 系统的概念

#### 1. 系统

系统（System）的概念是信息系统基础概念之一，也是我们常用的词汇。一般来说，系统是由一些元素组成的，这些元素之间存在着密切的联系，通过这些联系达到某种目标。因而系统也可以说是为了达到某种目标而相互联系的元素集合。

通常，系统被认为是一个整体，它是由若干个具有独立功能的元素（Element）组成的，这些元素之间互相联系、互相制约，共同完成系统的总目标。

美国国家标准协会（ANSI）对系统的定义是：各种方法、过程或技术结合在一起，按一定的规律相互作用，以构成一个有机的整体。

国际标准化组织技术委员会（ISO/TC）对系统的定义是：能完成一组特定功能的，由人、机器及各种方法构成的有机集合体。

美国《韦氏（Webster）大辞典》中，系统被解释为：有组织的或被组织化的整体；结合着的整体所形成的各种概念和原理的结合；由有规则的相互作用、相互依存的形式组成的诸要素的集合。

我国著名科学家钱学森认为，我们把极其复杂的研制对象称为“系统”，即由相互作用和相互依赖的若干组成部分结合成的具有特定功能的有机整体，而且这个“系统”本身又是它所从属的更大系统的组成部分。

根据上述定义，可以认为客观世界都是系统。例如，一个细胞是一个系统，一个生物个体是一个系统，一个生物群体也是一个系统；一个气体分子是一个系统，大气



层也是一个系统；一个班级是一个系统，一个企业是一个系统，一个社会组织也是一个系统。系统分层分类，有小有大，小到基本粒子，大到地球世界、太阳系、银河系乃至整个宇宙。

## 2. 系统结构

任何一个系统都具有一定的结构，否则不成为系统。所谓系统结构（Architecture），是指系统内各元素之间物理上或逻辑上的关系，如各元素在数量上的比例关系，时间上的先后关系，空间上的连接关系，人与人之间的隶属关系、血缘关系、同志关系，等等。系统内各元素之间的关系有些是静态稳定的，有些是动态变化的。

## 3. 系统功能

系统的功能（Function），即系统要达到的目标或要发挥的作用，是系统的基本属性。不同的系统一般具有不同的系统功能，但从本质上讲，系统的功能就是接受物质、能量与信息，并进行变换，产生并输出另一种形式的物质、能量与信息。

由此又可以说，系统就是按照某种结构把其元素组织起来的具有某种整体功能的一个统一体。

通常，称系统中有意义的元素为实体（Entity），描述实体特征的变量称为属性（Attribute），实体运动的行为称为活动（Activity），描述在任何时间的形态的变量称为状态变量（State）。

一个大的系统往往比较复杂，常常可按其复杂程度分解成一系列小的系统，这些小系统叫做包含它的大系统的子（分）系统，也就是说，这些子（分）系统有机地组成了大的系统。

图 1.1 显示了一个典型的系统组成图。这个系统的目的是将蛋糕原料烘制成蛋糕。系统边界定义了该系统与其他系统的区别（环境）。框内的数字显示了系统元素。这些元素包括原料（元素 1、2、3、4）、搅拌（元素 5）、用于加热烤箱的能量（元素 6）和烤箱（元素 7）。元素 1~4 及 6 作为输入，元素 5 和 7 是处理，最后的蛋糕是输出。

图 1.1 中有两个子系统，标记为子系统 A 和子系统 B。例如，子系统 A 是搅拌原料的物理处理，子系统 B 是烘制。通常，人们关心系统内的特殊元素，许多时候，这些特殊元素可以表示为更小的元素或子系统，例如，图 1.1 中的元素 7（烘箱）是一个用于加热的机械系统，即其本身又是一个系统。

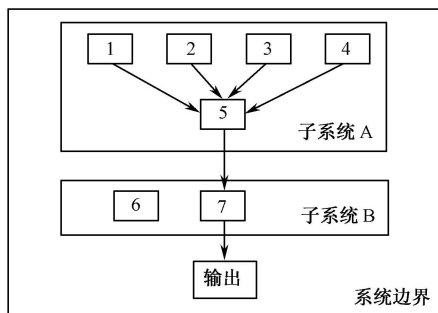


图 1.1 系统组成示例

### 1.1.2 系统的特性

#### 1. 目的性

任何系统都有目的和目标。我们建立一个系统，就是为某个目标服务的，每个系统都有其要达到的目的和应完成的任务或功能。例如，教育系统的目的是为了提<sub>高</sub>教学水平、提高人才的素质；又如，企业经营管理系统的目的可能是，在市场需求的基础上，根据生产的特点，在限定的资源和组织结构的相互协调下，完成生产任务，达到规定的质量、成本和利润等各项指标；再如，一个武器系统的目的可能是，满足各种战术要求，实现各种技术指标，达到给定的性能、经济与生产指标。

系统的目的决定着系统的基本作用和功能，并通过系统的功能达到和实现。系统的功能往往又通过一系列子系统的功能来体现，但这些子系统的目标之间往往互相有矛盾，其解决的办法是在矛盾的子目标之间寻求平衡和折中，以求达到总目标的最优。

建设一个新系统的第一步是确定系统的目标，这个目标必须是明确的、切合实际的，切忌提出含糊、空洞、脱离实际可能的目标。

#### 2. 相关性

系统的组成要素是相互依存又相互制约的，子系统之间也是如此。例如，在国民经济系统中，工业系统为农业系统提供机械设备、化肥等，而农业系统为工业系统提供原料、粮食和市场等。

系统组成要素之间的相互作用和约束一定要合理、协调和容易控制。因此，在划分子系统时，既要有适当的相对独立性、降低相关性，又不要分得过细。

### 3. 结构的层次性

系统的层次性是指系统的每个元素本身又可看作一个系统，即系统可分为一系列的子系统，这种分解实质上是系统目标的分解和系统功能、任务的分解，而各子系统又可分解为更低一层的子系统。例如，某信息系统组织的结构可以表示为图 1.2 所示的形式。

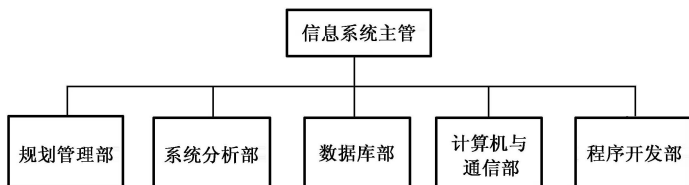


图 1.2 某信息系统组织的结构图

### 4. 整体性

组成系统的各个元素不是简单地结合在一起，而是有机地组成一个整体。每个元素都要服从整体，追求整体最优，而不是每个元素最优。评价一个系统时，不要只从系统的单独部分，即系统的元素或子系统来评价，而要从整个系统出发，从总目标、总要求出发来评价。只有当系统的各个组成部分和它们之间的联系服从系统的整体目标和要求、服从系统的整体功能并协调地活动时，这些活动的总和才能形成系统的有机活动。这样，系统的功能可望发挥各子系统功能之和的几倍、甚至几十倍。换句话说，系统概念就是“全局”观点。

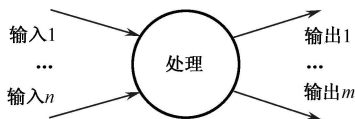
### 5. 环境适应性

一个系统本身总是从属于更大的系统，它是这个大系统的一个子系统。任何系统都存在于一定的环境之中，环境可以理解为一个系统的补集。系统总要受到环境的影响和制约，系统也要对环境的变化做出某种反应。把环境对系统的影响称为激励或冲击，而把系统对环境的反应称为响应。系统要想发挥它应当的作用，达到应有的目标，就一定要适应环境的要求。

## 1.1.3 系统的一般模型

系统可以是物理的，也可以是抽象的。抽象系统一般是概念、思想或观念的有序集合。物理系统不仅局限在概念范畴，还表现为活动或行为。一个实际的物理系统的

模型从宏观上看有输入、处理和输出三部分，如图 1.3 所示。



### 1. 系统输入

系统接受的物质、能量和信息称作系统的输入（Input），亦称外部激励（Stimulus）。

### 2. 系统输出

系统经变换后产生的另一种形态的物质、能量和信息称作系统的输出（Output），亦即对每个激励做出的响应（Response）。

### 3. 系统的环境

系统的环境（Environment）是为系统提供输入或接受它的输出的场所，即与系统发生作用而又不包括在系统内的其他事物的总和，简称外部环境或环境。

### 4. 系统的边界

系统的边界（Boundary）是指一个系统区别于环境或另一个系统的界限。有了系统的边界，就可以把系统从所处的环境中分离出来。可以说，系统的边界由定义和描述一个系统的一些特征来形成。边界之内是系统，边界之外是环境。

有系统就必有其边界。一般来说，系统边界的划分一方面既要使边界包含系统的元素、结构及目标所共同涉及的范围，另一方面又要在满足系统目标的前提下，使边界包含的内容尽可能少，直至仅包含那些保证完成系统目标的最少的必要部分。

作为一个系统，一般应具备三个独立的特征：

- ① 有元素及其结构；
- ② 一定的目标；
- ③ 确定的边界。

系统可能是简单的，也可能是复杂的。工厂、医院、商店和银行等都可被看作是系统。这些系统的目标是获取最大利益或顾客满意度。这些系统的元素包括机器、工人、管理人员，其输入是劳动、资本、土地和设备等，输出是产品和服务。表 1.1 显

示了一些系统的目标、输入和输出。

表 1.1  系统及其目标、输入、输出

系统	目标	元  素		
		输入	处理	输出
学校	获得知识	学生、教师、管理者、书本、设备等	讲授、研究、服务等	接受教育的学生、方法研究、社会服务等
医院	健康	医生、病人、护士、设备等	诊断、吃药、检查等	健康

按照系统的一般模型来认识、理解和处理系统时，可先将系统视为一个黑盒子 (Black Box)，按系统的输入和输出研究其外部特性，然后随着认识程度的深入，不断地展开黑盒子的内容，使其逐渐由“黑”变“灰”、变“透明”，直至“完全透明”。

系统的一般模型还可以扩展为若干系统（或子系统）相连接的情况，图 1.4所示是两个系统串联的情况，图 1.5所示是子系统层次展开的情况，图 1.6所示是子系统一般连接的情况。



图 1.4  系统连接模型 1

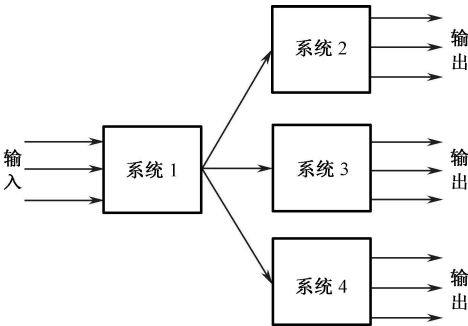


图 1.5  系统连接模型 2

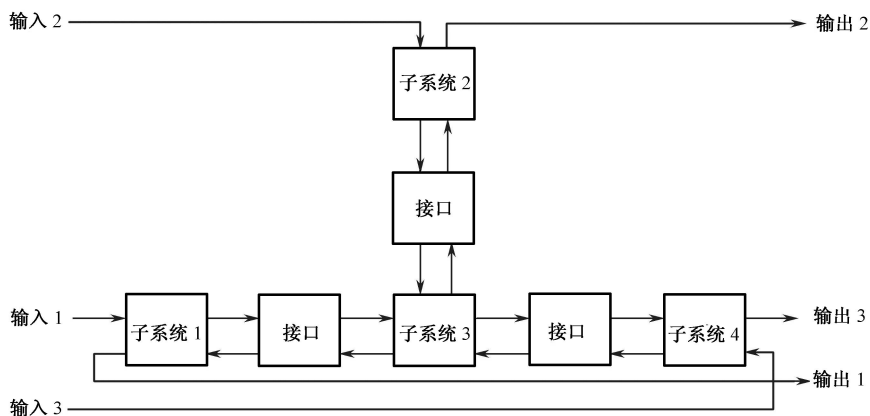


图 1.6 系统连接模型 3

### 1.1.4 系统层次与系统分类

#### 1. 系统层次

E. E. Boulding 提出了一般系统理论的系统层次的概念。以客观世界为出发点，把物理界、生物界及社会界的所有系统分为三类九个层次，并以此作为系统的基本运行单元，如图 1.7 所示。

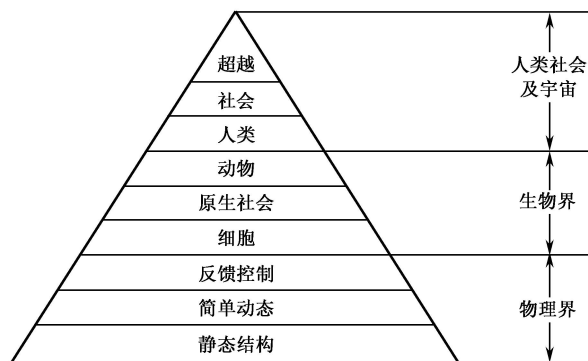


图 1.7 系统层次

第一层是静态结构系统，如宇宙、地球等系统。物理上讨论的简单力学系统及数学上讨论的代数等所描述的系统，大致都属于这一层次的系统，只有准确地描述了系

统的静态结构，才有可能进一步做出动态的分析与研究。

第二层是简单动态系统。这些系统的基本行为具有预先决定的、必然的运动方式，如太阳系中各行星的活动规律、钟摆的摆动规律等。微分方程是描述该类系统的常用工具与方法。

第三层是反馈控制系统。它与上述的第一、第二层的主要区别在于其系统内还具有传递及处理信息的能力，并有反馈的功能。导弹系统是这一层次中的标准系统。控制理论是描述该类系统的一个方法。没有人参加的信息系统也属于这个层次。

第四层是细胞系统。它具有自我维持能力的功能，与环境具有明显的物质、能量与信息的交流，是开放系统中最基础的结构形式或单元。与前三层比较，该层增加了一个需要适应环境的功能。

第五层是原生社会系统。这类系统的结构具有将类似的部件加以组织，并使之具有承担不同功能分工的能力。典型的例子是植物，如植物的根、茎、叶的细胞，在分裂初期，它们作为系统元素其结构在本质上并无区别，但因其位置及执行功能的差异，这些元素的功能也逐渐分工或特定化。

第六层是动物系统。该系统比植物系统增加了移动性的目的行为及自我觉察的能力。动物系统具有更强的凭借眼、耳等器官吸收信息的能力，而且同时具有以大脑为中心的神经系统。它有将吸收的信息转化为意识的组织者的作用，从而具有对接受的刺激做出反应的能力。越是高等的动物，其行为越不是对某一特定刺激的反应，而是对意识结构或对整体系统环境的反应。因此，该类系统的行为比较难以预测，这主要是由于意识参与了刺激与反应的过程所造成的。

第七层是人类系统。把个人作为一个系统，除了具备动物系统所具有的全部特征外，人还具有自我意识。这与动物的自我觉察有着本质的不同，它已跃升到具有语言和记忆的境地，从而能对外界的信息具有接受、解释、创造记号及变换等能力。而动物系统的符号仅是作为警告性的反应。

第八层是人类社会系统。该系统由其他的系统层次加以组合而成，它关心信息的内涵和意义、系统的价值程度、人类情绪的表现等。大多数有人参加的信息系统都属于这一层。

第九层是超越系统。这是当前尚不可知的系统。宇宙是否还可能发展出比地球上的人类社会系统更高一层的超越系统，还有待探讨。

## 2. 系统分类

从不同的角度出发，系统可以有各种各样的分类。一般将系统分类如下。

### (1) 自然系统与人工系统

自然系统的组成部分是自然物质，其特点是自然形成的，如生物系统、植物系统、

原子核结构系统、气象系统等。

人工系统是为了达到人类需求的目的，由人所建立起来的系统，如生产、交通、经营管理、经济、运输等系统。人工系统种类繁多，一般可归纳为三种类型：一是由人将零、部件装配成工具、仪器和设备，以及由它们所组成的工程系统；二是由一定的制度、组织、程序、手续等所组成的管理系统和社会系统；三是根据人对自然现象和社会现象的科学认识而建立起来的科学体系和技术体系。

实际上，大多数系统是自然系统与人工系统相结合的复合系统。在人工系统中，许多是人们运用科学力量，认识、改造了的自然系统。随着科学技术的发展，会出现越来越多的人工系统。了解自然系统的形成及其规律，是发展和创造更多的人工系统的基础。以后所讨论的系统一般都是指人工系统。

### （2）实体系统与概念系统

实体系统的组成要素是具有实体的物质。例如，由机械、矿物、生物等所组成的系统。概念系统是由概念、原理、方法、制度、程序、步骤等非物质实体所组成的系统，如科学技术系统、管理系统、教育系统等。

在实际生活中，实体系统和概念系统在多数情况下是结合的，概念系统为实体系统提供指导和服务，而实体系统是概念系统的服务对象。例如，计算机管理信息系统中的计算机和数据是实体系统，而管理的程序、方法等组成概念系统。

### （3）开放系统与封闭系统

开放系统的特点是系统与外界环境之间有物质、能量和信息的交换。封闭系统则与此相反，它与外界环境之间不存在物质、能量和信息的交换。但是从系统的思想观点来看，几乎一切系统都是开放系统，即使是过去物理学、机械学、热力学系统中占主导地位的所谓孤立系统（即封闭系统），也可视为开放系统的一种极端的特例（与外界的物质、能量和信息的交换都等于零）。为了明确一个系统的性质（开放系统或封闭系统），必须知道它与环境之间有无物质、能量和信息的交换。因此，必须首先确定系统边界，研究边界上物质、能量和信息的交换情况。一般地说，封闭系统具有刚性的、不可贯穿的边界，而开放系统的边界则具有可渗透性。生物、物理系统（都属于实体系统）的边界比较容易确定，社会、经济和概念系统的边界则往往较难确定。

开放系统的运动规律性的趋势是走向稳定和有序，表现为系统内部可用能的不断增加、可用能指标熵（Entropy）值的减少。这是系统与环境相互作用的结果。系统内部产生的熵可以向外转移，系统也可以从环境中吸取能量和组织性，使系统内部的组织化程度提高，有序化趋向增强。封闭系统则与外界环境完全没有物质、能量和信息的交换，系统内部的摩擦损耗使可用能不断减少，熵值不断增加，系统内部组织化程度越来越低，最后达到熵极大值时，系统的运动和发展停滞，相当于死亡状态。因此，



任何系统必须是开放型的才能使系统走向组织化、有序化。

#### (4) 静态系统与动态系统

动态系统中系统的状态变量是时间的函数，即描述其特征的状态变量是随时间而变化的。静态系统则是表征系统运动规律的数学模型中不含有时间因素的系统，即模型中的变量不随时间而变化。静态系统只是动态系统的一种极限状态，即处于稳态的系统。在实际工作中，以分析和研究动态系统为主要目的。

此外，系统还可以分为线性系统与非线性系统、确定系统与随机系统、适应系统与非适应系统等。具体的系统可能千变万化，但基本上可以看作是由上述各种系统的交叉组合形成的。图 1.8 所示已经勾画出一个完整清晰的系统分类的轮廓。

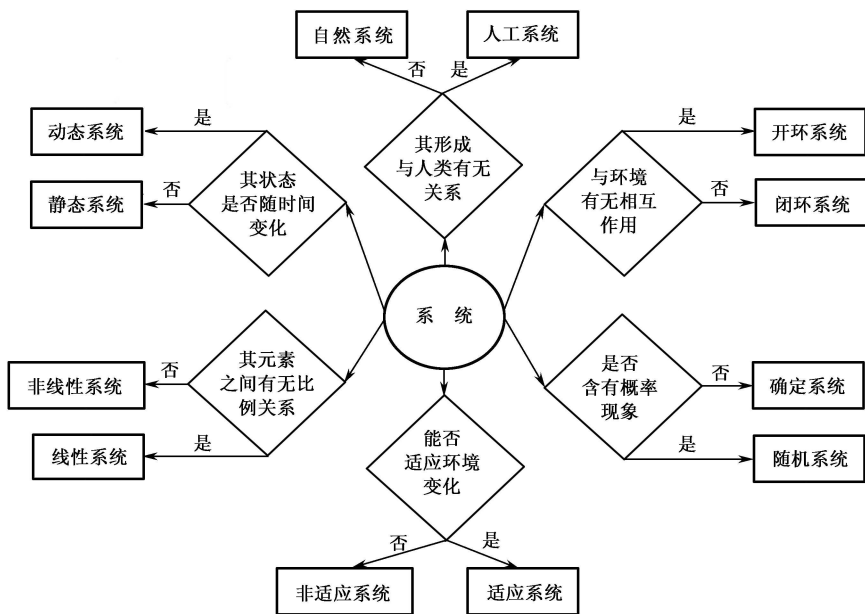


图 1.8 系统分类

### 1.1.5 系统学基本原理

#### 1. 整体性原理

元素或子系统按一定的结构组成系统后，便产生了它们在分别独立作用时所没有的新属性。所以，系统的属性个数总是多于各个元素在分别独立作用时的分属性的个数的和。系统目标函数的值与这些分属性对目标函数的贡献效应之和之间，存在大于、相等及小于三种关系，具体是何种关系，取决于系统结构及元素属性的本质。这就是系统的整体性原理。

设系统由  $n$  个元素构成, 用  $A_i$  表示第  $i$  个元素独立作用时具有的所有分属性的集合 ( $i = 1, 2, \dots, n$ ), 用  $A$  表示系统的全部属性的集合, 则对任何系统, 有

$$A \supset \bigcup_{i=1}^n A_i$$

其中,  $A$  中多于各元素分属性和的那些系统属性是各个元素按系统结构连接在一起后共同作用的结果。例如, 人是由细胞组成的系统, 会制造和使用工具, 尽管细胞本身也很复杂, 但任何一个单独的细胞都没有也不可能有这样的属性 (即制造和使用工具)。又如, 阿波罗载人宇宙飞船及其运载火箭和地面测控设施是一个复杂的大系统, 能成功地把人送上月球并安全返回, 这样的属性是任何一个单独的部件或任何一个单独的人都不具备的。

属性是一个较为庞杂的定性的概念, 不便于作定量的描述。由于希望达到系统的目标是系统存在的全部意义, 故可考虑把系统的所有属性向系统的目标函数方向量化或投影, 即把每个属性对系统目标函数 (或功能) 的作用和影响量化成某种可以度量和累加的代数量, 称为投影量, 换句话说, 就是用目标函数值的单位去度量每个属性在这个量上的作用大小。若某属性对系统目标函数值起负面作用或影响, 则该属性在目标上的投影量为负; 若某属性对系统目标函数值起正面作用或影响, 则其投影量为正; 若某属性对系统的目标函数值无任何作用或影响, 则其投影量为零。这样, 就可以评价每个属性对系统目标函数的贡献大小了。

假定系统的目标函数值的大小为  $f_A$ , 系统中第  $i$  个元素独立作用时的所有属性关于  $f_A$  的全部投影量的代数和为  $a_i$ 。客观事实表明, 系统目标函数值  $f_A$  与该系统中所有  $n$  个元素分别独立作用时的投影量之和有如下三种可能的关系:

$$(1) f_A > \sum_{i=1}^n a_i;$$

$$(2) f_A = \sum_{i=1}^n a_i;$$

$$(3) f_A < \sum_{i=1}^n a_i。$$

第一个式子表明“整体大于部分之和”。整体大于部分之和的基本力量来自系统内各子系统 (元素) 的分工与协同效应。有人用“ $1+1>2$ ”来形象地表达该协同效应。协同 (Synergy) 现象是随处可见的。例如, 一个经营有方的企业的经济效益显然会超过其中的生产销售和财务等部门单干的经济效益的总和。

第二个式子表明“整体等于部分之和”。这种关系人们最熟悉, 也认识得最早。例如, 一个国家的人口等于各地方的人口之和, 整体的质量等于各部分的质量之和等。

第三个式子表明“整体小于部分之和”。这种关系往往容易被人忽视, 但实际上却

客观存在。例如，中国俗话说“一个和尚挑水吃，两个和尚抬水吃，三个和尚没水吃”。和尚运水的效果是可以累加的，而“没水吃”的结果说明三个人的整体功能不仅低于三个人独立挑水之和的功能，而且低于单独每个个人的独立功能。

系统整体与其部分和之间之所以存在上述三种关系，主要原因是系统结构带来了组合效应，这是各元素单独作用时所没有的。这些组合效应对系统目标函数来说可能是正面的协同效应，也可能是负面的消耗效应，也可能无任何效应。

系统的整体性原理揭示了系统作为一个客观实体存在的重要意义，对人类能动地改造世界有着极大的指导作用。

## 2. 最优化原理

系统结构的演进受系统目标的控制和系统环境的影响，服从一个统一的自然规律，即在保证实现环境允许系统达到的目标（功能）的前提下，使整个系统对时间、空间、物质、能量及信息的利用率最高。这就是系统的最优化原理。最优化原理指明了系统结构演进的方向。

系统的结构是完成系统目标（功能）的基础。例如，只有把计算机的各个芯片及零件设备按设计的结构装配起来，才能组成一个计算机系统，并具备计算的功能。如果只是把这些芯片和零件设备混乱地放在一起或胡乱地装配在一起，则显然不具备计算的功能，因而也不是一个计算机系统。

不同的结构产生不同的功能和性能。一个有相同元素的系统，如果其结构不同，则功能一般也不同。例如，计算机网络是一个信息系统，在单个计算机系统功能与台数都相同的条件下，对于可靠度目标函数来说，全互连分布式网络的可靠性一般都高于星形网、环形网等网络的可靠性。又如，如果一个信息系统的单个模块数及其独立功能都不变，仅改变部分或全部模块之间的调用关系，即改变系统结构，则常常会产生不同的系统功能。一个企业或一个公司，如果人员、设备都不改变，仅改变其管理及运作体制，如采用优化劳动组合，引入竞争、监督及保障机制，即改变系统结构，则往往会提高企业或公司的效益。

既然系统结构对系统目标功能有如此大的影响，那么对于既定的系统来说，系统内各元素到底按照什么样的原则进行分工、协作，才能使系统的功能与性能最好呢？系统最优化原理正是回答这一问题的。只要在许可的条件下，系统的时间、空间、物质、能量及信息等五个利用率尚未达到最高，那么该系统内分工和协作的方式（即结构）就不会稳定，就一定要从落后的结构向先进的结构发展，直到许可的条件下五个利用率达到最高为止。这就是最优化原理阐明的自然界发展从无序走向有序的一条必然规律。

### 3. 木桶原理

系统技术水平的高低，不仅仅取决于构成系统的各个元素的技术水平的高低，而且，更取决于系统整体技术水平的高低。这就是常说的“木桶”原理。

木桶的装水量，不是取决于桶壁中的长板，而是取决于桶壁中的短板。这一点非常重要，对工程技术和系统技术改造等工作都具有重要的指导意义。是通过增强系统中的强项来提高系统的整体性能，还是通过增强弱项，特别是迅速添补“缺项”来提高和改善系统的整体性能呢？答案毋庸置疑。

需要补充的是，系统中各个要素的地位和作用是不相同的。在一个系统中，各个要素不是平等关系，而是各自占有不同的地位和起到不同的作用。例如，木桶的“底板”和木桶的“提把”，是组成木桶的两个元素，虽然看起来有把没底的桶，比有底没把的桶，在外形上更像桶，但是，有底没把的桶具备装水的功能，是真正的桶，只是不太好用，而有把无底的桶则不成为桶。

### 4. 模型与模拟化原理

由于系统之间的相似性，所以从某个系统中总结出的规律，可以推广和还原到与它相似的系统上去，这一原则称作模型与模拟化原理。这是具体研究系统的一个方法性原则。

模型是对相应的真实对象和真实关系中那些有用的和令人感兴趣的特性的抽象化。因此，模型描述可视为是对与真实世界中的物体或过程相关的信息进行形式化的结果。模拟就是在模型上做实验。

模型可以用一个六元组表示： $M = \{O, G, T, V, R, S\}$ 。

其中，O 表示模型的对象集；

G 表示模型的目标集；

T 表示模型系统所处的环境及约束条件集；

V 表示模型的变量集，包括内部变量、外部变量及状态变量；

R 表示模型变量之间的关系集；

S 表示模型的状态集，从初态到终态。

真实世界是复杂和动态的。基于这个事实，研究客观世界的方法是：针对人与外部世界的相互作用，在科学的基础上建立问题空间的“形式”模型，用这一形式模型来反映和描述所要解决的真实问题。科学研究的绝大部分工作就是实现对问题的形式化描述和建立模型。例如，自然科学是通过加大对大自然的观察和试验，总结、提炼出对客观事物的抽象表示方法和定律。这些方法和定律是已被证实的对事物表述的“形式”模型。

人类认识世界和改造世界的过程首先是建立模型和分析模型，然后根据分析的结论去指导人类的行动。

(1) 建立模型：通过对客观事物建立一种抽象的表示方法，来表征事物并获得对事物本身的理解，从而建立现实世界的模型。

(2) 分析模型：依据模型进行计算，求解验证；通过模型的考察建立对客观事物的分析结论。

模型与模拟化方法的基本步骤是：

- (1) 为系统建立模型；
- (2) 对模型进行实验与研究；
- (3) 将对模型研究的结果推广到与该模型相似的原型系统上。

系统模型是指将面向系统目标的系统的信息集合起来，并与系统有相似的物理属性、逻辑属性或数学描述的实体。原来的系统则称为模型的原型。为原型系统建立模型的过程常常简称为建模。

建模必须遵守如下 6 个基本原则。

(1) 相似性。模型与所研究系统在属性上应具有相似的特性和变化规律，也就是说，“原型”与“替身”之间具有相似的物理属性或数学描述。

(2) 切题性。模型只应该针对与研究目的有关的方面，而不是一切方面。这是因为对于同一个系统，其模型不是唯一的，模型的选择应针对研究目的。

(3) 吻合性。模型结构的选择，应尽可能对可利用的数据作合理的描述。通常，其实验数据应尽可能由模型来解释。

(4) 可辨识性。模型结构必须选择可辨识的形式。若一个结构具有无法估计的参数，则此结构就无实用价值。

(5) 简单性。从实用的观点来看，由于在模型的建立过程中忽略了一些次要因素和某些非可测变量的影响，因此实际上的模型已是一个被简化了的近似模型。一般而言，在实用的前提下，模型越简单越好。

(6) 综合精度。综合精度是模型框架、结构和参数集合等项精度的一种综合指标。若有限的信息限制了模型的精度，则最有效的模型就应是各方面精度的平衡和折中。

若上述原则间出现冲突，则要寻找合理的折中，但特定的折中方案却依赖于模型的对象，因而没有固定的程序。

建模的基本步骤如下。

(1) 明确目标。明确目标即明确以下问题：为什么样的系统建立模型；该系统需要建立怎样的模型；怎样和原型系统比较。

- (2) 明确系统边界和约束条件。
- (3) 确定构成系统的最小功能单元或元素。
- (4) 确定主要因素和主要变量，即面向系统目标抓主要矛盾。
- (5) 明确输入与输出及其他各种关系。
- (6) 规定符号和代号。
- (7) 建立逻辑关系图或数学模型等。

模拟的方法也就是实验的方法。由于它把实验观察法与逻辑抽象法的特点结合在一起，因此能有效地突破时间、空间的限制。

系统的存在离不开环境，所以，任何模拟都应考虑与环境的关系。通常有下列 3 种方法。

- (1) 用系统模型结合真实环境进行模拟。如宇宙航行中用动物作为生理模型代替人去探险。
- (2) 用真实系统结合环境模型进行模拟。如军事演习，参战部队是真实系统，而演习的环境则是战争环境的模型。
- (3) 用系统模型结合环境模型进行模拟。如飞机模型在风洞中“飞行”。

### 1.1.6 系统性能和标准

#### 1. 效率和有效性

效率 (Efficiency) 和有效性 (Effectiveness) 是评价系统性能的两个方法。

效率是关于生产与消耗之比的度量，其范围是  $0\% \sim 100\%$ 。例如，马达的效率是能量生成与能量消耗之比，一些马达的效率低于  $50\%$ ，因为摩擦和生热造成了能量消耗。

效率是用于比较系统的相对的标准。例如，公司财务部门引入财务自动化系统后比原先的手工劳动更有效率，这是因为财务自动化系统更节省人力，计算更加迅速、准确，工作强度也大大降低。

有效性用于衡量系统完成目标的程度。可以这样计算它：实际完成的目标与预计总目标之比。例如，一个公司的目标是降低成本 10 万元，为此引入了一个新的控制系统并安装使用，希望帮助完成这个目标。然而，使用后发现最终仅降低成本 8.5 万元，那么这个控制系统的有效性就是  $85\%$ 。

与效率相似，有效性也是用于比较系统的相对的标准。

## 2. 系统性能的评价

对于一个信息系统来说，有时很难确定系统的效率和有效性。通常，判断系统的好坏可以由以下 4 点作定性观察。

(1) 目标明确：每一个系统都有一个目标。这个目标选择得是否合适、明确，是评价系统好坏的主要方面。

(2) 结构清晰：一个系统可以由若干子系统组成，子系统还可以划分为更小的子系统。结构清晰是指这种层次关系及其内部联系便于实现系统的目标，且条理清楚、信息流畅。

(3) 联系清楚：指上述联系通过定义清楚的接口进行。

(4) 能观能控：系统与外界有清楚的界面，外界可以通过输入控制系统的行为，又可以通过输出观察系统的行为。

评价系统性能也称为系统性能标准的使用。

系统性能指标一旦确定，就将作为评价和确定系统全部性能的标准。

## 1.2 信息

信息是当今社会的标志。随着社会的进步，人们越来越认识到知识就是力量，信息就是财富。信息在社会生产和人类生活中起到越来越大的作用，并以其不断扩展的内涵和外延，渗透到社会、经济和科学技术的众多领域，使人类继工业社会之后，正式迈入信息社会。信息的增长速度和利用程度，已成为现代社会文明和科技进步的重要标志。

### 1.2.1 信息的概念

在 19 世纪以前，人们对信息的认识一直处于原始和经验阶段。虽然理论上没有深刻认识，但是人类对信息的利用却从来没有停止过，尽管这种利用是低水平的，常常是不自觉的，而且基本上是通过人类的信息器官的天赋功能来进行的。即便如此，人类还是创造了不少方法来利用信息。例如，创造文字来记录信息，发明纸张、印刷术存储信息，发明算盘进行信息处理等。人类真正自觉认识信息问题并触及信息本质是在 20 世纪 40 年代以后。理论上的主要标志是申农 (C. E. Shannon) 1948 年发表的论文《通信的数学理论》和维纳 (Wiener, N.) 的专著《控制论——动物和机器中的通信和控制问题》，并由此奠定了信息科学的基础。此后，信息科学蓬勃发展，极大地推



动了社会的进步。

什么是信息？也有人认为，信息就是消息，是具有新内容、新知识的消息。也有人认为，信息就是情报，是对人们有价值的情报。历史上，关于信息的定义有几十种之多，我们无须去研究哪种定义更加确切，但是关于信息有以下两点可以明确。

（1）信息的存在不以主体（如人、生物或机器系统）的存在为转移，即使主体根本不存在，信息也可以存在，它在客观上反映某一事物的现实情况。

例如，人们使用照相机、摄像机可以记录一些正在发生的事件，尽管当时的真实场景现在可能已经不存在，但是这些设备记录下来的信息却可以展现当时的情景。再如，考古学从古代遗留的信息中了解曾经发生的历史事件，这些事件在人类历史中或人类出现以前就已经存在，只不过没有为现在的人所感知和利用而已。目前，信息已经被公认为物质世界的基本属性之一，它与物质和能量构成物质世界的三大支柱。从这种意义上讲，信息就是信息，它既不是物质也不是能量，它是关于物质运动千差万别的状态的知识，是事物内部结构和外部联系的运动状态和方式。

（2）信息在主观上可以接受和利用，并指导人们的行动。人类在改造客观世界的过程中，需要从客观世界中获取信息，通过感觉器官感知信息，通过大脑分析、处理信息。这时，信息就具有了比本体意义更为丰富的内涵。作为主体的人，能够理解信息的状态和方式，能够判断信息的效用价值，能够利用信息来创造价值。从这种意义上讲，信息是主体所感知或该主体所表述的相应事物的运动状态及其变化方式，包括状态及其变化方式的形式、含义及效用。

信息（Information）与数据（Data）是信息系统中最基本的术语，两者关系密切，但含义并不相同。

信息系统工程中对数据的理解是：记载下来的事实，客观实体属性的值。或者说，数据是可以记录、通信和识别的符号，它通过有意义的组合来表达现实世界中实体（具体对象、事件、状态或活动）的特征。数据的记载方式多种多样，在逻辑上，数据主要包括数值型、文字型、语音型、图形图像型、视频型等多种类型。数据用什么形式表达，取决于不同的媒体。

信息系统工程中对信息的理解是：

- （1）信息是表现事物特征的一种普遍形式；
- （2）信息是数据加工的结果；
- （3）信息是数据的含义，数据是信息的载体；
- （4）信息是帮助人们做出决策的知识；
- （5）信息是由实体、属性、值所构成的三元组。

我们可以这样来理解信息：信息是构成一定含义的一组数据。这个提法把信息理



解为一组有意义的数据，从而对信息处理的理解就更清楚一些。信息不是一般的数据，而是一种已经被加工为特定形式的数据。这种数据形式对接收者来说是有确定意义的，对人们当前和未来的活动产生影响并具有实际价值。

由上述可知，数据与信息有着不可分割的联系。信息是由处理系统加工过的数据，是一种原料与成品的关系，如图 1.9 所示。

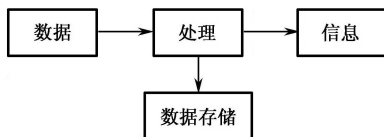


图 1.9 数据与信息的关系

### 1.2.2 信息的性质

#### 1. 事实性

信息最早是“关于客观事物的可通信的知识，通信把信息用于事实”。所以，事实是信息的中心价值，不符合事实的信息不仅没有价值，而且可能其价值为负。例如，战场上，获取敌方的情报越准确，则战胜对方的把握越大，己方的损失越小，否则损失越大。

信息反映了客观事物的运动状态和方式，但是信息不是客观事物本身。信息可以脱离其源事物而相对独立存在。

事实性是信息的第一和基本的性质。在信息系统中，应当充分重视信息的事实性，破坏信息的事实性必将给管理、决策带来错误。事实性是收集信息时最应当注意的性质。

#### 2. 等级性

信息系统是分等级的，对于同一问题，处于不同的管理层次，所要求的信息不同。同样，信息具有等级性，它和管理层一样，分为战略级、管理级和操作级。不同等级的信息其性质也不同。

例如，从来源上说，企业战略规划信息，即关于企业的方向、目标等，关于企业之间的联合，关于世界市场的开拓等，多来自外部，并且其寿命最长，保密程度最高，处理方法最不固定，信息精度要求最低；管理信息，如企业资源配置、新产品品种、生产效益等，在来源上有内有外，其保存寿命、保密要求等较战略规划信息低；操作

信息，如生产各项指标完成的情况、考勤信息等，大多来自企业内部，其保存寿命、保密要求最低，信息的处理方法往往是固定的。

### 3. 可压缩性

信息可以压缩，进行集中、综合和概括，而不至于丢失信息的本质。例如，关于牛顿第二定理的论述可以用一个简单的公式表示，实验时可以把实验得到的大量数据组成一个经验公式等。信息压缩过程中要丢掉一些信息，但丢掉的应当是无用的或不重要的信息。

无用的信息主要有两种，即干扰信息和冗余信息。干扰信息，如杂波信号，本来就应当滤掉；冗余信息则要视具体情况而定，例如，检、纠错码在通信中是必要的，但存入信息系统中时通常无须保留。

由于计算机处理速度、通信容量、存储容量等的限制，所以有时没有能力收集、处理和存储一个事物的全部信息，有时也没有必要收集、处理和存储一个事物的全部信息，这时，就要对信息作一定的分析，根据管理的目标提取与目标相关的信息，舍弃其他信息。这就是信息的不完全性。例如，在视频信息传输时，通常要对信息作压缩，但必须保证图像的可识别性和真实性。只有正确地舍弃信息，才能正确地使用信息。

### 4. 扩散性

信息的扩散是其本性，它总是力图冲破保密的非自然约束，通过各种渠道和手段向四面八方传播。信息的密度越大，信息源和接收者间的梯度越大，信息的扩散性越强。

在信息的扩散存在两面性，一方面它有利于知识的传播，另一方面可能造成信息的贬值。在信息系统中，要特别注意信息的保密问题，如果没有很好的安全、保密手段，就不能保证用户正确使用信息系统，从而导致信息系统的失败。

### 5. 可传递性

信息可以通过多种传输渠道、采用多种传输方式进行传递。信息传递需要借助于物质载体。传输渠道可以是报纸、书籍、无线电广播、电话，也可以通过计算机网络和卫星等进行传输。实现信息传递功能的物质载体的形式多种多样，称为信息媒介。信息传递的过程同时伴随着物质、能量的传递，但它较物质、能量的传递更加优越。信息的可传递性和易于传递性，加快了信息资源的传输，加快了社会的发展。

## 6. 共享性

信息的共享性是其重要性质。信息可以被共同接收，共同占有，共同享用。物质交换原则是，一方得到一物，另一方必然失去一物。但信息交换双方不仅不会因信息交换使其中一方失去信息，而且会增加新的信息。这种非零和的特性，造成了信息共享的复杂性。

信息的共享性有利于信息成为组织的一种资源。严格地说，只有达到信息共享，信息才真正成为资源。

## 7. 价值性

信息是经过加工的、有意义的数据，是一种资源，因而是有价值的。索取一份情报，或者利用大型数据库查阅文献所付的费用，是信息价值的体现。信息的使用价值必须经过转换才能得到，况且，如果信息“衰老”很快，则转换必须及时，否则信息就没有什么价值了。

信息又是可以增值的，在积累的基础上，信息的增值可能从量变到质变。例如，每天天气预报的信息，预报期一过就不再有用，但多年天气预报信息的积累，可能会使我们发现气候变化的规律，从而指导我们的生产和生活。

信息的增值性、再生性使我们能在信息废品中提炼有用的信息，在司空见惯的信息中分析出重要的趋势。

### 1.2.3 信息运动模型

信息的运动与物质和能量的运动不同，它有自己的特点。由于信息既具有客观性，又具有主观性，所以信息成为客观物质世界和主观精神世界的桥梁，显然，信息的运动比物质和能量的运动要复杂得多。在人类认识和改造客观世界的过程中，形成了众多的学科门类，尽管当时人们没有认识到是信息在起作用，但是，这种不自觉的行为已经为人类带来了巨大的进步。

认识信息的运动，不能只考虑信息运动的客观方面而脱离了主观性，否则缺乏对信息的再升华，从而导致对客观世界认识的僵化，失去创造性，毕竟信息只有为人类主观认识后才能更好地改造世界。同时，也不能孤立地只考虑主观方面而脱离了客观性，否则信息缺乏存在的物质基础，容易蜕变为形而上学的唯心主义。因此，应当从信息的客观性出发，将两者有机结合起来，以更好地认识和利用信息，为人类服务。

信息运动大致可分为两个方面，即本体论意义上的信息运动，它反映对象（事物）

运动的状态和方式，是客观方面；认识论意义上的信息运动，它反映由主体所发出（表述）的主体思维运动的状态和方式（代表主体意志），是主观方面。

图 1.10 所示是信息运动的一种模型，它反映了信息从客体进入主体，经过在主体中的运动再作用于客体的运动过程。

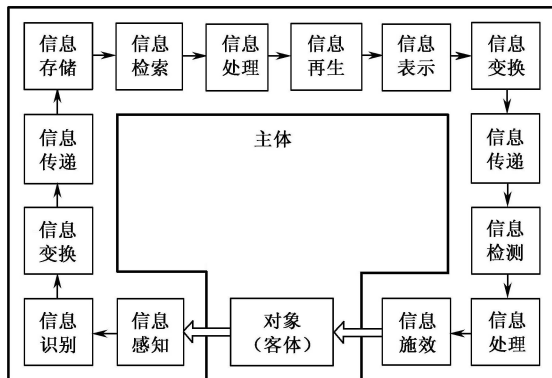


图 1.10 信息运动模型

这个模型概括了以下重要的过程单元。

(1) 信息感知：通过感知器官获取外部世界的事物信息，完成本体论意义的信息向认识论意义的信息的转变。

(2) 信息识别：对感知的信息加以辨识和分类。

(3) 信息变换：将识别出的信息进行适当形式的转换（一般是变换它的载体）。

(4) 信息传递：将信息由空间的某一点转移到另一点。

(5) 信息存储：收到信息后以适当的形式进行存储。

(6) 信息检索：当需要信息时，把存储者的信息迅速、准确地提取出来。

(7) 信息处理：为便于使用，需要对信息进行适当的加工处理，包括分析、比较和运算等。

(8) 信息再生：在信息处理的基础上就可能获得关于对象运动的规律性认识（即再生出更为本质的信息），并形成针对客体对象的策略。

(9) 信息表示：将主体再生的信息用适当的方式表示出来。

(10) 信息变换：对再生信息进行适当变换并以一定的方式表现出来。

(11) 信息传递：把加工变换的再生信息从空间的某一位置（主体所在处）转移到另一位置（客体所在处）。

(12) 信息检测：信息在传递过程中可能受到噪声等因素的影响，需要把再生信息从干扰的背景中分离出来。

(13) 信息处理：为便于再生信息发挥作用，还需要对其进行适当的加工。

(14) 信息施效：运用再生信息对客体对象的运动状态和方式进行调整。

从图 1.10 可以看出，信息的运动过程就是不断地了解、控制对象，使它逐渐地由初始的运动状态和方式转移到最终的运动状态和方式。只有当上述所有过程都发挥正常作用，主体才能从本体论意义上的信息中提取认识论意义上的信息，并从中形成有关客体对象的正确认识，在这个基础上再生出反映主体意志的信息，并通过它的反作用实现对客体的变革。

在处理、再生和施效的过程中，主体应当具有智能，而且，智能水平越高，相应的信息过程就越有效，反之则越差。因此，智能活动是主体认识、改造世界过程的基本特征。

人利用信息的基本过程，就是获取客体的语法、语义和语用信息，经过与目标信息的比较、决策形成指令信息，最后经过控制和调整重新作用于客体的过程。这个过程是一个反馈控制过程，如图 1.11 所示。

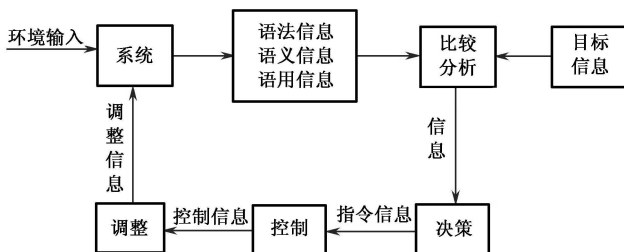


图 1.11 人利用信息的基本过程

## 1.2.4 信息的度量

### 1. 语法信息的度量

在语言学中，把语言中“词和词之间的结合方式”称为语法，我们借用这个术语，把只考虑“状态及状态变化方式”的信息称为语法信息。语法信息是事物运动及其变化方式的外在形式，不涉及这些形式的含义和效用，只考虑表示符号和符号之间的结合方式所包含的信息。语法信息是信息问题最基本的层次，研究信息度量的问题也是首先从语法信息开始的，而且目前也最为成功。

早在申农建立信息论以前，哈特莱（Hartley）在“信息传输”（1928 年）一文中提出了信息度量的一种方法，并指出了信息数量的大小仅与发信者在字母表中对字母的选择方式相关，而与信息的语义无关，并给出了信息度量公式：

$$I = \log S^n = n \log S$$

其中,  $S$  是字母表中字母的数目;  $n$  是每个消息的字母数目。申农接受了哈特莱关于信息的形式化思想, 并把他的信息度量推广到更有意义的情况。

申农对信息的研究基于一个简单的通信模型, 即消息传递系统。它由信源(发信者)、信道(消息传递的物理通道)和信宿(收信者)组成。消息从信源发出, 经过信道传输, 最后由信宿接收。所谓消息, 是指文字、语音、图像等这些能够为人民的感觉器官所感知的物理现象, 是把客观物质运动和主观思维活动的状态表达出来而构成的。信息附载在消息上, 信息是消息的内容, 消息是信息的具体反映形式。从通信的观点出发, 构成消息须具备两个条件: 一是能为通信双方理解; 二是可以传递。

在消息传递过程中, 收信者在收到消息以前不知道消息的具体内容(否则消息传递就没有意义)。对收信者来说, 是一个从不知到知, 从不确定到确定的过程。从通信的过程来看, 收信者的所谓“不知”就是不知道发信者将发送描述何种运动状态的消息。例如, 打电报时, 报文“母病愈”是母亲身体健康状态的一种描述, 而母亲的身体健康情况会表现出不同的状态。可见收信者在看到报文以前存在“不确定性”, 他不能确定母亲的健康状况如何。一旦看到报文以后, 只要报文是清楚的, 在传递过程中没有出现错误, 那么他原来的“不确定性”就没有了。所以通信过程是一种从不确定到确定的过程, 是消除不确定的过程。不确定消除了, 收信者就获得了信息。显然, 信息的测度与不确定性的程度有关, 从数学角度看, 不确定性就是随机性, 其大小与概率有关。

这样, 申农找到了“形式化”和“概率论”两个工具, 并在此基础上创立了信息论。

### (1) 不定度

不定度是关于事物运行方式和状态的不确定的程度。由概率可知, 经常发生的事件, 其出现的概率较大, 关于它的不定度较小, 反之亦然。如果事件  $x$  以  $P(x)$  的概率出现, 则关于事件  $x$  出现的不定度定义为

$$H(x) = \log(1/P(x)) = -\log P(x)$$

申农指出, 采用对数函数形式具有以下优点。

- ① 比较实用。工程中的重要参量, 如时间、带宽等与可能状态的数目成线性关系。
- ② 比较直观。人们经常直观地用与公共标准进行线性比较的方法来度量事物, 如两张磁盘的信息存储量将两倍于一张的存储量等。
- ③ 数学上比较合适。许多极限运算采用对数比较简便。

## (2) 信息量

既然信息量与不定度有关,那么接收者消除了某事件的不定度后,就可以衡量他获得的信息量的大小。假定接收者在观察随机事件  $x$  之前,事件  $x$  所具有的不定度为  $H(x)$ ,而通过观察事件  $x$  之后,不定度被完全消除,此时,关于事件  $x$  的不定度为零,那么他就获得了  $H(x)$  大小的信息量。关于  $x$  的信息量,定义如下:

$$I(x) = H(x) - 0 = H(x) = -\log P(x)$$

式中,由于没有考虑后验概率的情况,所以也称  $I(x)$  为自信息。

如果考虑后验概率的情况,则接收者收到事件  $y$  后,对事件  $x$  尚存的不确定性的信息量大小,称为互信息。互信息与后验概率  $P(x|y)$  有关,定义为:

$$I(x|y) = \log(1/P(x|y)) = -\log P(x|y)$$

信息量采用什么测度单位,取决于对数以什么为底。如果取以 2 为底,则所得到的信息量单位称为比特 (binary unit, bit);若取以 e 为底的自然对数,则得到的信息量单位称为奈特 (nature unit, nit);若取以 10 为底,则所得信息量单位称为迪特 (digital unit, dit)。

**【例 1.1】** 设一幅图像由约 78 万个像素点组成,且每个像素点有 10 级亮度和 10 个色彩 (如 VGA 彩屏,就是  $1\,024 \times 768 = 78.643\,2$  万个像素点,256 个色彩)。求该图像所含的信息量。

由于 78 万个像素点中的每个像素点都有 10 级亮度乘以 10 个色彩的组合可能,故全部可能的图像是  $(10 \times 10)^{78 \times 10\,000}$  个。假设所有可能的图像以等概率出现,则某一幅图像  $x$  出现的概率  $P(x) = 100^{-78 \times 10\,000}$ 。把它代入信息量计算公式可得一幅图像  $x$  的信息量:

$$I(x) = -\log_2 100^{-78 \times 10\,000} \text{ bit} \approx 78 \times 10^4 \times 6.64 \text{ bit} \approx 5.18 \times 10^6 \text{ bit}$$

**【例 1.2】** 设播音员的用语总数为 1 万条,求播音员所播的 1 000 条语句中的信息量。

该播音员播出的 1 000 条语句中的每一条用语都有 10 000 种可能,故全部的可能语句组合是  $(10\,000)^{1\,000} = 10^{4\,000}$ ,于是特定的某 1 000 条语句事件  $y$  出现的概率  $P(y) = 10^{-4\,000}$ 。把它代入信息量计算公式得到 1 000 条语句  $y$  的信息量:

$$I(y) = -\log_2 10^{-4\,000} \text{ bit} \approx 1.3 \times 10^4 \text{ bit}$$

## (3) 信息熵

对于某一随机事件,当它有多种可能性时,为描述其整体的信息量,采用概率论中数学期望的方法来定义。

设某一随机事件  $X$ ,其结果不确定,它的多种可能结果为  $x_1, x_2, x_3, \dots, x_n$ ,每种结果出现的概率分别为  $p_1, p_2, p_3, \dots, p_n$ ,则事件  $X$  的概率空间描述为

$$S = \begin{Bmatrix} X \\ P \end{Bmatrix} = \begin{Bmatrix} x_1, x_2, x_3, \cdots, x_n \\ p_1, p_2, p_3, \cdots, p_n \end{Bmatrix}$$

其中,  $0 < p_i < 1, \sum_{i=1}^n p_i = 1$ 。

事件  $X$  的平均信息量为

$$H(X) = K \sum_{i=1}^n p(x_i) \frac{1}{\log p(x_i)} = -K \sum_{i=1}^n p(x_i) \log p(x_i)$$

上式与物理学中熵的计算公式只差一个负号, 因此, 可以把信息称为负熵, 即信息熵。规定  $0 \log 0 = 0$ , 即概率为零的事件, 信息量为零。

上式中  $K$  为系数, 其值与不同的单位制有关。当对数取以 2 为底, 且  $n = 2$ ,  $p(x_1) = p(x_2) = 0.5$  时, 令

$$H(X) = -K \sum_{i=1}^n p(x_i) \log_2 p(x_i) = 1$$

则有  $K = 1$ 。  $H(X)$  有 1 bit 的信息量, 即含有两个独立的等概率可能状态的随机事件具有的不确定性被全部消除所需要的信息量。

信息熵的性质如下。

① 对称性。当变量  $p_1, p_2, \cdots, p_n$  的顺序任意互换时, 熵不变, 即

$$H(p_1, p_2, \cdots, p_n) = H(p_2, p_3, \cdots, p_n, p_1) = H(p_n, p_1, p_2, \cdots, p_{n-1})$$

该性质说明熵只与随机变量的总体结构有关, 与信源的总体的统计结构有关。如果某些信源的统计特性相同 (含有的符号数和概率分布相同), 那么, 这些信源的熵就相同。

② 确定性。

$$H(1, 0) = H(1, 0, 0) = \cdots = H(1, 0, 0, \cdots, 0) = 0$$

该性质意味着, 如果从总体来看, 信源虽然有不同的输出, 但它只有一种情况必然发生, 而其他情况几乎都不可能出现, 那么这个信源是一个确定信源, 其熵等于零。

③ 扩展性。

$$\lim_{\epsilon \rightarrow 0} H(p_1, p_2, \cdots, p_q - \epsilon, \epsilon) = H(p_1, p_2, \cdots, p_q)$$

该性质说明, 当信源取值数增多时, 若这些取值数对应的概率很小 (接近于零), 则信源的熵不变。虽然小概率事件出现后, 给予接收者较多的信息, 但从总体考虑时, 因为这种小概率事件几乎不发生, 所以它在熵的计算中占的比重很小。这也是熵的总体平均性质的一种体现。

④ 非负性。非负性即  $H(X) \geq 0$ 。



该性质显然成立。由于  $0 < p_i < 1$ , 所以当对数底大于 1 时,  $\log(p_i) < 0$ , 而  $-\log(p_i) > 0$ 。这种非负性对于离散信源的熵是适合的, 但对连续信源这一性质并不存在。

⑤ 可加性。可加性即统计独立信源  $X$  和  $Y$  的联合信源的熵等于分别熵之和。

$$H(XY) = H(X) + H(Y)$$

⑥ 极值性。极值性即等概率分布时, 熵达到最大。

$$H(p_1, p_2, \dots, p_n) \leq H\left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}\right) = \log(n)$$

极值性表明, 等概率分布的信源其平均不确定性最大, 称为最大离散熵定理。由此, 在实际生活中, 几支实力相当的球队进行比赛, 肯定最精彩。

**【例 1.3】** 对常用汉字 (如国标一级字库中有 3 755 个汉字) 分别进行使用频度统计, 然后加权求和, 得到每个汉字的平均信息量:

$$E(\text{汉字}) \approx 9 \text{ bit}$$

**【例 1.4】** 对常用英语中出现的字母分别进行使用频度统计, 得到字母使用概率分布表, 见表 1.2。

表 1.2 字母使用概率分布表

字母	空格	E	T	O	A	N	I	R	S
概率	0.195	0.075	0.072	0.064 5	0.063	0.059	0.055	0.054	0.052
字母	H	D	L	C	F	U	M	P	Y
概率	0.047	0.035	0.029	0.023	0.022 5	0.022 5	0.021	0.017 5	0.012
字母	W	G	B	V	K	X	J	Q	Z
概率	0.012	0.011	0.010 5	0.003	0.003	0.002	0.001	0.001	0.001

以随机变量  $x$  表示英文信源, 以  $x_i$  表示英文中的 26 个字母和空格, 并用  $P(x_i)$  表示各个字母或空格出现的概率, 则可得英文信源的平均信息量:

$$H(X) = - \sum_{i=1}^{27} p(x_i) \log_2 p(x_i) \approx 4.02 \text{ bit}$$

## 2. 语义信息的度量

语法信息量只是表明了主体关于事物运动及其变化方式的外在形式方面存在的不确定性消除了多少, 但是, 认知主体在获得信息时, 不仅要知道“是什么形式”, 而且要了解“是什么意思”, 也就是说, 主体要获得信息的意义。所谓语义信息, 就是事物运动状态和方式的含义。

不难看出, 语义信息的度量是一个复杂的问题。因为它涉及语法符号的含义、上下文关系、语言环境的变化及认知主体的知识结构等诸多因素。自 20 世纪 60 年代以

来, 人们就开始语义信息问题的定量研究, 并取得了一定进展, 但是尚未得到很好的解决方法。

为了描述语义信息, 首先必须建立“含义”的表征方法。可以采用逻辑真实度的方法来描述。于是, 设立一个“状态逻辑真实度”参量, 记为  $t, 0 \leq t \leq 1$ , 其定义为

$$t = \begin{cases} 1 & \text{状态逻辑为真} \\ 1/2 & \text{状态逻辑不定} \\ \alpha \in (0, 1) & \text{状态逻辑模糊} \\ 0 & \text{状态逻辑为伪} \end{cases}$$

若随机事件  $X$  的状态为  $X = \{x_1, x_2, \dots, x_n\}$ , 各状态的概率为  $P = \{p_1, p_2, \dots, p_n\}$ , 各状态的逻辑真实度为  $T = \{t_1, t_2, \dots, t_n\}$ , 则事件  $X$  的语义信息结构为:

$$S = \begin{Bmatrix} X \\ T \\ P \end{Bmatrix} = \begin{Bmatrix} x_1, x_2, x_3, \dots, x_n \\ t_1, t_2, t_3, \dots, t_n \\ p_1, p_2, p_3, \dots, p_n \end{Bmatrix}$$

语义度量公式如下:

$$H(X, T) = K \sum_{i=1}^n t_i p(x_i) \frac{1}{\log p(x_i)} = -K \sum_{i=1}^n t_i p(x_i) \log p(x_i)$$

### 3. 语用信息的度量

尽管语义信息较语法信息的含义更为深刻和广泛, 但是主体在获取信息时, 更加关心它“有什么用处”, 即信息的效用问题。借用语言学中的术语, 把关于“事物状态及状态改变方式的效用”的信息称为语用信息。显然, 度量语用信息是一个更加复杂的问题。信源发出信息以后, 其效用因人、因时、因地而不同, 同一信息作用于不同对象和在不同的环境下, 其效用不同, 甚至完全相反。

与在语义信息的度量中引入状态逻辑真实度的方式类似, 采用效用度的概念来处理状态效用的表征问题。设状态效用度的参量为  $u$ , 它满足:

$$u = \begin{cases} 1 & \text{状态效用最大} \\ b \in (0, 1) & \text{状态效用模糊} \\ 0 & \text{状态效用最小} \end{cases}$$

若随机事件  $X$  的状态为  $X = \{x_1, x_2, \dots, x_n\}$ , 各状态的概率为  $P = \{p_1, p_2, \dots, p_n\}$ , 各状态的效用度为  $U = \{u_1, u_2, \dots, u_n\}$ , 则事件  $X$  的语用信息结构为

$$S = \left\{ \begin{matrix} X \\ U \\ P \end{matrix} \right\} = \left\{ \begin{matrix} x_1, x_2, x_3, \cdots, x_n \\ u_1, u_2, u_3, \cdots, u_n \\ p_1, p_2, p_3, \cdots, p_n \end{matrix} \right\}$$

语用度量公式如下：

$$H(X, U) = K \sum_{i=1}^n u_i p(x_i) \frac{1}{\log p(x_i)} = -K \sum_{i=1}^n u_i p(x_i) \log p(x_i)$$

#### 4. 信息与有序

在热力学中，熵用来描述系统的紊乱程度。热力学第二定律告诉我们，存在耗散的封闭系统，有序的运动状态向无序的运动状态转变，最终达到热平衡。随着时间的推移，热熵总在不断地增大，这就是热力学熵增加的原理。

一个完全无序、处于热平衡状态的事物无疑是简单的事物，一个具有高度对称的有序状态的事物，如完美的晶体也可看作是简单的事物。一般地讲，描述简单事物的状态需要的信息量小，而描述复杂事物的状态所需要的信息量多。因此，复杂事物的状态处于完全有序与完全无序之间。

任何信息都可由文字、语音和图像来描述，而这些又都可以录制在磁带上，由数字串来代表，像计算机用二进制数字串那样。一个完全随机的没有任何规律的数字串和一个完全规则的如(10101010...)数字串所带的信息量都是很少的。对前者只要说它是随机的，对后者也只要说它是10的重复就够了。而一个既有确定规则、中间又杂以随机的数字串才带有更多的信息。规则的部分反映的是必然性，随机的部分反映的则是偶然性。因此，一个包含信息量大的复杂系统的状态是由必然性和偶然性结合而产生的。例如，人类社会的历史就是由社会发展的必然规律和一些偶然的因素等共同决定的。

由“信息是事物的运动状态和方式”的定义可知，如果战场上的一支军队对敌方的兵力部署、武器装备、战斗力强弱、行动方向、后勤补给乃至天时、地利及己方的各种情况等整个战争的态势（状态和方式）即信息掌握得准确充分，那么加上正确的指挥，这支军队一般会立于不败之地。同样，如果一个工厂对产品销售，能源、材料供给，生产、质量、库存及效益等状态和方式即信息掌握得准确充分，那么加上正确的决策，这个工厂一般会兴旺发达。反之，如果对敌方、对市场一无所知或知之甚少，则军队必然一败涂地，工厂必然一片混乱。这说明，可用“信息”来描述系统的有序程度，关于某个系统的必要信息量掌握得越多，则该系统的结构就越有规则，系统就越有序。系统内部要素越混乱，它们联合起来共同完成有意义工作的能力就越小。换言之，系统的总能量可以分为用来完成有意义工作的能量（有用能量）和杂乱运动的

能量（无用能量）两部分。随着熵的增加，在系统总能量不变的情况下，无用能量不断增加，有用能量不断减少。如果封闭系统得不到外界的能量补充，则当熵趋于极大值时有用能量逐渐完全丧失，系统实际上就“死亡”了。开放系统虽然也具有熵增加的倾向，但由于它不断接受来自环境的输入和制造返回环境的输出，所以在这种新陈代谢的过程中其有用能量不断得到补充，无用能量不断受到抑制，能始终充满活力。换句话说，系统从环境获得的输入中有一部分实际上被系统用来抑制熵的增加，这一部分可以称为负熵（Negative Entropy）。开放系统正是由于得到不断注入的负熵才避免了熵不断增加而失去工作能力的结局的。因此，我们又把信息称为负熵，它增进有序。

1.2.5 信息的分类

与其他事物的分类问题一样，信息分类也有许多种方法，可以把信息分为表 1.3 所示的一些类型，它有助于我们对信息的理解。

表 1.3 信息分类

分类角度	分 类
认识层次	语法信息、语义信息、语用信息
观察的过程	实在信息、先验信息、实得信息
信息的地位	客观信息、主观信息
信息的作用	有用信息、无用信息、干扰信息
信息的逻辑意义	真实信息、虚假信息、不定信息
传递方向	前馈信息、反馈信息
发生领域	宇宙信息、自然信息、社会信息、思维信息
应用领域	工业信息、农业信息、军事信息、政治信息、科技信息、文化信息等
信息源的性质	语音信息、图像信息、文字信息、计算信息等
信号的形式	连续信息、离散信息、半连续信息等

1.2.6 信息的特点与价值

1. 信息的特点

信息存在许多有趣的特点或属性。

(1) 信息没有质量

信息有别于物质。一部《孙子兵法》可载于一车竹简，也可载于一本书，还可缩微于几平方厘米的胶片或存于一张光盘。信息载体（或媒体）的改变一点儿不影响信息的内涵，而且有助于信息的传播。

## （2）信息容易复制

信息有别于能量。能量虽然守恒，但一定形式的能量，只会使用一点少一点。信息则不然，将信息复制或传播给别人，自己一点也不丧失，信息也不改变。信息可以共享。所以，维纳说：“信息就是信息，既不是物质，也不是能量。”

## （3）信息取之不竭

信息是事物运动的状态和方式。事物的运动是永恒不息的，故信息永不枯竭，不会出现像材料和能源短缺一样的现象。

## （4）信息需要载体

任何信息都必须依附在其载体上，才能存储和传播。如前所述，数据是信息的逻辑载体。除了一般常用的数据外，还有像人类语言、文化习惯等高度浓缩的数据，记载着人类或民族发展进步的历史信息。客观物质是信息（或数据）的物理载体。存储信息需要物理载体，如绳结、石子、竹简、陶器、丝绸、纸张、磁盘、光盘、脱氧核糖核酸（DNA）及人脑等；传播信息也需要物理载体，如声波、电缆及光纤等。

## （5）信息超越时空

信息可自由地超越时间和空间进行传播。在地球范围内乃至浩瀚宇宙间的通信，是信息在超越空间。阅读历史，以古鉴今，是信息在超越时间。信息在传播过程中，有的会因“噪声”而变质，有的被存储，有的自行消亡。

# 2. 信息的价值

众所周知，没有材料（或物质流），制造机器如同空中楼阁；没有动力（或能量流），机器如同废铁；没有信息（或信息流），机器如同野马失去控制。物质、能量、信息三位一体，共同构成了客观世界。人类的一切生产、生活活动都是物质、能量与信息组合的过程。信息是一种极为重要的资源，因而也具有极为重大的价值。

## （1）信息支持决策

所谓决策，就是把收集到的信息与要求的目标信息进行比较分析，选择最合理的对策进行实施，并随时监督实施，依据实施反馈的新信息调整对策。简单地说，决策就是为了达到行为目标而采取某种对策的过程。管理需要决策，指挥需要决策，任何有目的的行动都需要决策，而任何决策都必须先有信息。如果没有及时和适用的信息，则决策必然是瞎指挥、乱拍板，后果不堪设想。一方面，收集和分析信息需要付出社会劳动；另一方面，准确、及时与适用的信息会导致正确的决策，进而取得效益，因此信息是具有价值的。

## （2）信息构成知识

知识是人类在不断认识自然与改造自然的过程中逐步积累的关于世界的客观描述，

是信息的积累和提炼。正确地运用知识会产生巨大的力量，从而推动事物的发展和进步。随着时代的前进和科学技术的发展，我们面临的许多问题往往都比过去更复杂，而问题的复杂度又往往与该问题相关的知识难度成正比。所以，如果具备了解决该问题的有关信息及知识，就为最终解决该问题奠定了基础。从寻找和收集信息的角度看，可以说，哪里有知识，哪里有决策，哪里就有信息。而如果从使用信息的角度看，又可以说，哪里需要知识，哪里需要决策，哪里就需要信息。

### （3）信息增进有序

信息普遍存在于自然界和人类社会，它是生物界、人类社会和人造系统赖以生存和发展的重要资源。任何系统物质和能量的变化、运动、交换一般都以信息为先导，并受信息的控制，如生物个体的发育受脱氧核糖核酸中遗传信息的控制和影响，候鸟的迁徙、鱼类的回游、爬虫的冬眠和惊蛰等无不受信息的控制。人类社会更是这样，一切生产和生活的活动都是在人类发出的信息的控制下进行的，只不过人类对信息不像生物那样简单、被动地受控，而是具有记忆和处理的能力，能进行逻辑推理和形象思维，建立新概念，发现新规律，并进而改造客观环境，使之适合于人类生活的需要。

总之，由熵增原理可知：一个孤立封闭的系统最终会走向无序，要改变这种状况或促进系统走向有序，就必须使系统向环境开放，并从外部环境中输入负熵，进而使整个系统的熵减少，系统朝确定的方向演进。所以，信息反映了系统的确定程度，信息增进了系统的有序发展。

## 1.2.7 信息与管理

信息与管理有着密切的联系。信息是管理的纽带，信息的获取、加工和传递是管理活动的基础，信息活跃在整个管理过程的各个环节；管理就是通过信息进行管理，管理过程就是信息沟通的过程。科学的管理必须建立在良好的信息管理的基础上，而信息技术为科学管理提供了重要的技术基础。

在管理过程中，管理者的主要工作是信息沟通。管理学家明茨伯格（Henry Mintzberg）把管理任务归纳为三大任务和十大角色，如图 1.12 所示。

管理者是负责一个组织或组织的一个下属单位的人。所有管理者被授予一个组织单位的一个正式职权，他们的地位确立在正式职权的基础上。这种地位产生了各种人际关系，它们之间互相提供信息以便做出决策。管理者的基本任务是设计和维护一种环境，使身处其间的人们能够在组织内协调工作，从而有效地完成组织目标。

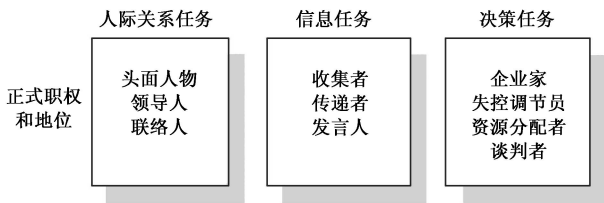


图 1.12 管理任务

(1) 人际关系任务：保证组织的正常运转。头面人物代表组织主持、参加各种社交应酬活动。领导人领导下属完成工作，并使下属的要求与组织的目标相配合。联络人保持组织内部和外部的联络与沟通。

(2) 信息任务：保证信息的集散畅通。管理者，作为信息收集者，必须时刻了解不断变化的环境，通过联络网以不同的方式了解情况；作为传递者，必须与组织的其他成员共享大部分信息；作为发言人，向上级或外界介绍组织的情况。

(3) 决策任务：保证决策的正确及时。管理者，作为企业家要勇于开拓、不断创新，制定实施战略计划和行动方案；作为调节员，必须对出现的各种问题及时进行处理和解决；作为资源分配者，必须对组织的资源等进行合理调配；作为谈判者，要代表组织与外界谈判，同时，对复杂问题与组织成员进行讨论和磋商。

可见，在上述三类任务中，信息沟通起着至关重要的作用。

管理学作为一门独立的学科，其理论的发展一般可分为四个阶段。

第一个阶段是从 19 世纪末到 20 世纪初形成的古典管理理论。以泰勒 (F. W. Taylor)、法约尔 (Henri Fayol) 和韦伯 (Max Weber) 等人为代表，主张用科学的精神、科学的态度和科学的方法进行管理，从而形成比较完整的古典理论体系，它们主要包括科学管理原理、管理过程理论及行政组织理论等。

第二个阶段是从 20 世纪 20 年代开始的人际关系理论及 50~60 年代的行为科学理论。它们是随着经济的发展，劳动者受教育程度的提高，人们需求的多样化，管理的重点转向人的因素，继而研究工作环境、人际关系和工作效率的关系的基础之上发展起来的。

第三个阶段是从 20 世纪 50 年代末发展起来的现代微观管理理论。在此期间，管理学全面引进了系统论、信息论、控制论、运筹学、经济数学、计算机科学技术和社会学、人类学、政治学等其他科学理论，形成了研究企业经营决策和系统优化管理的新兴综合性学科，产生了以巴纳德 (C. I. Barnard) 为代表的社会系统理论、以西蒙 (H. A. Simon) 为代表的决策理论和以布莱克特 (P. M. Blackett) 为代表的管理科学理论。



第四个阶段是从 20 世纪 70 年代以后不断扩展的宏观管理理论。管理研究进入了以战略管理为主的研究组织战略规划与环境关系的高层宏观管理时代，形成了以波特（M. E. Porter）为代表的竞争战略理论，以及以哈默（M. Hammer）为代表的企业再造理论。

## 1.3 信息系统

大千世界中存在着各式各样的系统，在任何一个系统中，其内部必然有物质、能量和信息的流动，其中信息控制着物质和能量的流动，使系统更加有序。从系统的观点看，信息流在整体上构成一个系统，因此，在任何复杂系统中都有一个沟通各子系统、各部门的信息系统作为它的一个子系统存在。信息系统的作用和其他子系统不同，它不从事某一具体功能、做某一具体工作，而是关系全局的协调一致。在部队里，上级下达的命令、指示、通知、通报等都是信息。下级向上级报告任务完成的情况、反映部队的情况也都是信息。对上级来说如果出现情况不明，对下级来说如果不理解上级的意图，则都可以说是信息系统不好。社会组织的种类和功能千差万别，但它们都有自己一定形式的信息系统，而且信息系统工作得好坏与整个组织的效益的关系极大，可以说信息系统是整个系统的神经系统。

### 1.3.1 信息系统的概念

广义上说，任何系统中信息流的总和都可视为信息系统（Information Systems, IS）。它们需要对信息进行获取、传递、加工、存储等处理工作，如生命信息系统。随着科学技术的进步，信息的处理越来越依赖于通信、计算机等现代化手段，使得以计算机为基础的信息系统得到了快速发展，极大地提高了人类开发、利用信息资源的能力。因此，目前普遍认同的信息系统是指基于计算机、通信网络等现代化的工具和手段，服务于管理领域的信息处理系统。它是 20 世纪中叶信息科学、计算机科学、管理科学、决策科学、系统科学、认知科学（Cognitive Science）及人工智能等相互渗透而发展起来的一门学科。

研究信息系统的主要任务是研究信息处理过程的内在规律，以及基于计算机等现代化手段的形式化表达和处理规律。同时，信息系统又是一门实践性很强的应用科学，在实践中产生，又在实践中不断发展。经过 50 多年的不断探索和实践，已初步形成了自己独具特色的理论和技术体系（尽管目前还不十分完善），其应用的触角已深入到社会生活的各个方面。以信息系统为中心的信息产业已成为当今信息化社会最活跃、最



有生机、最有潜力的支柱产业之一。

信息系统概念涉及的内涵非常广泛，目前尚无统一的定义，名称也不完全一致。例如，有人使用“管理信息系统”，有人使用“信息与决策系统”，也有人使用“组织的信息系统”等。

对信息系统概念的研究可以追溯到早期对电子数据处理系统和管理信息系统的概念的研究。“管理信息系统”一词最早出现在1970年，肯尼文（W. T. Kennevan）从管理的角度定义它为：以口头或书面的形式，在合适的时间向经理、职员及外界人员提供过去的、现在的、未来的有关企业内部及其环境的信息，以帮助他们进行决策。他强调信息支持决策，但没有包括计算机和应用模型。1985年，美国的戴维斯（G. B. Davis）给管理信息系统一个较完整的定义：它是一个利用计算机硬件和软件，手工作业，分析、计划、控制和决策模型及数据库的人机系统；它提供信息，支持企业或组织的运行、管理与决策功能。这个定义较全面地说明了管理信息系统的目标、功能和组成。

从系统的观点看，信息系统是对信息进行采集、处理、存储、管理、检索和传输，必要时能向有关人员提供有用信息的系统，如图1.13所示。

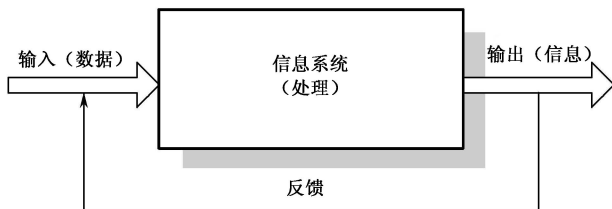


图 1.13 信息系统的定义

从上面的定义可知：

- (1) 信息系统的输入与输出类型明确，即输入是数据，输出是信息；
- (2) 信息系统输出的信息必定是有用的，即服务于信息系统的目标，它反映了信息系统的功能或目标；
- (3) 信息系统中，处理意味着转换或变换原始输入数据，使之成为可用的输出信息；处理也意味着计算、比较、交换或为将来的使用进行存储；
- (4) 信息系统中，反馈用于调整或改变输入或处理活动的输出，对于管理、决策者来说，反馈是进行有效控制的重要手段；
- (5) 计算机并不是信息系统所固有的。实际上，在计算机出现之前，信息系统就已经存在，如动物的神经信息系统。

信息系统可以由人工或计算机来完成，后者称为基于计算机的信息系统，也正是

我们研究的对象。许多信息系统多是先从手工开始，然后过渡到计算机化。但是，将一个人工信息系统简单地计算机化，是不能保证提高系统性能的。如果信息系统的基础有缺陷，那么使其计算机化可能扩大这些缺陷的影响。

### 1.3.2 信息系统的发展

人类自进入文明社会以来，一直从事信息处理工作。但是计算机的发展改变了人们几千年来传统观念，促使人们进一步研究信息处理、信息系统、信息资源充分利用的规律。这就是现代信息系统作为一门学科诞生和发展的基础。

20 世纪 50~60 年代，计算机应用导致了信息系统的诞生、发展，并带来了信息系统的首次繁荣。但随着时间的推移，它也逐步暴露出许多的局限性和不足，在实际应用中出现了波折，这引起了人们对信息、信息系统、信息处理规律的进一步思考，并使人们在探索中不断丰富和完善自己，推动信息系统的更大发展。

信息系统的发展经历了从电子数据处理阶段到决策支持系统阶段的发展过程。

在一个系统中，信息存在于系统的各个层次中。信息系统的设计，需要针对不同层次的管理人员，设定不同的功能目标。由 R. N. Anthony 提出的三级管理模型，将管理过程划分为三个层次：战略计划、管理控制、操作控制。在 Anthony 三级管理模型的基础上，R. V. Head 提出了信息系统的层次模型，如图 1.14 所示。

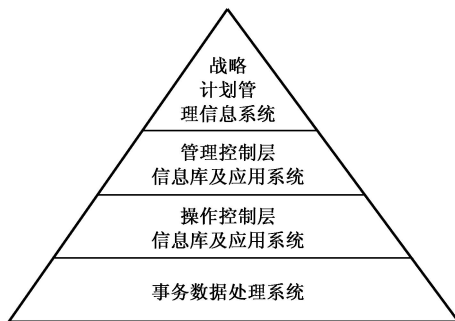


图 1.14 信息系统的层次模型

其中，操作控制层信息一般是用来显示天天要重复的操作过程的，通常利用事务数据处理模块、报表生成模块和查询模块来产生事务活动的单据、统计报表和查询应答。管理控制层根据战略计划层的要求，及时给出所需要的各种管理信息，这类信息通常带有统计或预测性质。这一层要求能为各级管理人员的管理活动，提供用于制定、组织、控制等活动的所需信息。战略计划层除了及时了解管理上所需的综合信息外，

有时, 需要使用各种数学模型和方法进行模拟和预测。这一层的信息系统不可能像操作控制层或管理控制层那样提供具体、详细的信息, 而且在管理方法上也复杂得多。通常, 是建立以数据库、模型库为基础的计算机决策支持系统。

信息系统除了辅助于不同层次的管理活动外, 还有一个重要的部分就是事务数据处理系统, 它为其余所有内部信息的辅助活动提供基础。因此, 整个信息系统的层次结构可用图 1.14 所示的金字塔来表示。其中, 金字塔的底部表示明确且结构化的规程和决策, 顶部代表着非结构化的处理和决策。

### 1. 数据处理系统

数据处理系统 (Data Processing System, DPS) 一般指用于操作层的重复出现、变化不大的各种过程处理和事务处理, 如工资计算、账务处理中的原始凭证录入等。这种系统多为一项一项地处理各种信息, 各项处理之间的联系很小。

DPS 是开发信息系统初级阶段的产物, 是建立下述各种信息系统的基础。

### 2. 管理信息系统

管理信息系统 (Management Information System, MIS) 是为实现系统的整体管理目标, 对各类管理信息进行系统、综合处理, 并辅助各级管理人员进行管理决策的信息处理系统。

MIS 主要由信息收集、信息存储、信息加工、人机对话与输出等部分及信息管理者所组成。MIS 一般可以根据决策层次、管理职能和信息处理方式, 分成若干个相互关联的子系统, 以便于整个系统的开发。

严格说来, MIS 只是一种辅助管理系统, 它所提供的信息需要由管理人员去分析、判断和决策。

### 3. 决策支持系统

管理决策的制定, 是一个包括确定目标、收集信息、探索方案及对各种方案进行分析、预测和选择的过程。而 MIS 往往只是按照它在建立时所确定的模式来收集、存储和加工信息。因此, 对于那些目标明确、具有确定的规则和程序及信息需求的决策问题即结构化决策, MIS 可以有效地支持决策中各个阶段的活动。但是, 现代管理决策中面临的问题, 往往是目标含糊不清, 多个目标相互冲突, 方案的比较和选取没有固定规则和程序可循, 所需信息不完整而且比较模糊的问题, 这类决策问题称为半结构化或非结构化决策问题。

决策支持系统 (Decision Support Systems, DSS) 是 MIS 的发展与深化。DSS 的

定义也很多，但通常具有以下特点。

- (1) 较强的语言处理和人机交互能力。
- (2) 由数据库、模型库和知识库组成的信息存储与管理系统。
- (3) 是将数学模型、算法和推理方法结合起来的问题处理系统。

#### 4. 计算机信息系统

DPS、MIS 和 DSS 都是以计算机为基础的信息系统，有时统称为计算机信息系统 (Computer Information System, CIS)。除了上述面向管理的计算机信息系统外，还有计算机辅助设计 (Computer Aided Design, CAD) 系统、计算机辅助制造 (Computer Aided Manufacturing, CAM) 系统、办公自动化 (Office Automation, OA) 系统、情报检索 (Information Retrieval System, IRS) 系统等。

总之，各种类型的以计算机为主要工具的信息系统正在不断发展，且在社会生活的各个领域发挥着越来越重要的作用。

### 1.3.3 信息系统的基本功能

信息系统是对信息进行采集、处理、存储、管理、检索和传输，必要时能向有关人员提供有用信息的系统。这个定义概括了信息系统的基本功能。

#### 1. 信息的采集

信息的采集即信息收集。信息系统必须首先把分布在各部门、各处和各点的有关信息收集起来，记录其数据，并转换成信息系统所需的形式。信息采集有许多方式和手段，如人工录入数据、网络获取数据、传感器自动采集等。对于不同的时间、地点和类型的数据，需要按照信息系统需要的格式进行转换，形成信息系统中可以互相交换和处理的形式，如传感器得到的传感信号需要转换成数字形式才能被计算机接收和识别。

信息采集是信息系统的重要环节。它关系到信息系统中流动和处理的信息的质量好坏，对信息系统的功能、作用和作用效果有着直接的影响。

#### 2. 信息的处理

信息的处理即对进入信息系统的数据进行加工处理，如对账务数据的统计、结算、预测分析等都需对大批采集录入到的数据作数学运算，从而得到管理所需的各种综合指标。信息处理的数学含义是：排序、分类、归并、查询、统计、预测、模拟及进行

各种数学运算。现代化的信息系统都是依靠规模大小不同的计算机来处理数据的，并且处理能力越来越强。

### 3. 信息的存储

数据被采集进入系统之后，经过加工处理，形成对管理有用的信息，然后由信息系统负责对这些信息进行存储保管。当组织结构相当庞大时，需存储的信息量是很大的，于是就必须依靠先进的存储技术。这时，有物理存储和数据的逻辑组织两个问题。物理存储是指将信息存储在适当的介质上；逻辑组织是指按信息的内在逻辑联系和使用方式，把大批的信息组织成合理的结构，它常依靠数据存储技术。

### 4. 信息的管理

一个系统中要处理和存储的信息量很大，如果不论重要与否、有无用处，盲目地采集和存储，则将成为数据垃圾箱。因此，对信息要加强管理。信息管理的主要内容是：规定应采集数据的种类、名称、代码等，规定应存储数据的存储介质、逻辑组织方式，规定数据传输的方式、保存时间等。

### 5. 信息的检索

存储在各种介质上的庞大数据要让使用者便于查询，这是指查询方法简便，易于掌握，响应速度满足要求。信息检索一般要用到数据库技术和方法，数据库的组织方式和检索方法决定了检索速度的快慢。

### 6. 信息的传输

从采集点采集到的数据要传送到处理中心，经加工处理后的信息要送到使用者手中，各部门要使用存储在处理中心的信息等，这些都涉及信息的传输问题，系统规模越大，传输问题越复杂。

## 1.3.4 信息系统的结构

按照系统的观点，可以把一个组织简单地看作是由三个子系统所组成的：管理子系统、执行子系统和信息子系统。信息系统除包含自身的管理外，还要将一般组织的管理功能融合进来。信息系统提供给执行子系统有效的信息，同时接受执行的结果，并将处理后的信息传输给管理子系统。这里讲的管理子系统与管理信息系统不同，管理子系统包括管理信息系统。

管理子系统根据组织的目标和方针，对执行子系统进行控制活动；执行子系统负责业务处理和物质交流等；信息子系统可以看作是插入上述两个子系统之间而负责收集、存储、处理和分发信息的功能部分，抽象地看，它包括信息库（模型库和数据库）和信息处理程序（管理控制程序、转换管理程序等）。三个子系统之间的关系如图 1.15 所示。

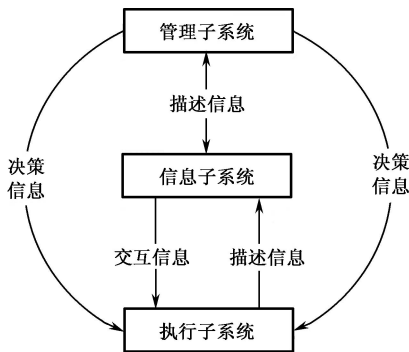


图 1.15 系统组织结构图

信息系统的结构是指各部件的构成框架，对部件的不同理解构成了不同的结构方式。

(1) 概念结构。从概念上看，信息系统由四大部件组成：信息源、信息处理器、信息用户和信息管理者。其中，信息源是信息的产生地；信息处理器负责信息的传输、加工、保存等；信息用户是信息的使用者，并利用信息进行决策；信息管理者负责信息系统的设计、实现和实现后的运行、协调。

(2) 功能结构。从使用的角度看，信息系统总是具有一个目标和多种功能，各种功能之间又有各种信息联系，从而构成一个有机结合的整体，形成一个功能结构。

(3) 软件结构。支持信息系统各种功能的软件系统或软件模块所组成的系统结构，是信息系统的软件结构。

(4) 硬件结构。信息系统的硬件结构是指硬件的组成及其连接方式和硬件所能达到的功能。

## 1. 信息系统的结构分类

### (1) 按处理功能的结构分类

按处理功能，信息系统的结构可分为如下五类：

① 事务数据处理功能

② 数据管理功能

- ③ 数据输入、输出功能
- ④ 数据通信传输功能
- ⑤ 数据分析预测功能

## (2) 按辅助管理层次的结构分类

按照 Anthony 提出的管理计划和控制活动的三级分类法, 信息系统又可按照其对管理活动的不同层次的辅助划分为三个层次的结构形式。

① 操作控制层: 通常利用事务数据处理模块、报表生成模块和查询处理模块来产生事务活动的单据、统计报表和查询应答。本层次的处理一般都是预先确定的, 较容易通过计算机来实现。

② 管理控制层: 该层要求能为企业各职能部门管理人员的管理活动, 提供用于衡量组织的成绩, 控制和组织生产活动, 制定、组织资源分配方案等所需的信息。

管理控制层信息系统的输出包括计划与预算、定期报告、特别报告、问题条件的分析、评审决策及查询应答等。

③ 战略控制层: 该层的处理较复杂, 需要采用模型、模拟方法等提供辅助, 需要对获得的数据进行深加工。其主要作用是: 提供组织当前能力的评价; 对组织未来的潜力预测; 分析外部环境因素; 估计竞争对手的能力; 提供备选方案; 对备选方案的资源规分类。

## (3) 按辅助管理活动职能的结构分类

① 对于企业, 可以划分为市场销售信息子系统、生产信息子系统、后勤信息子系统、财会信息子系统、人事信息子系统及其他职能信息子系统。

② 对政府综合经济部门, 有宏观综合计划控制信息子系统、统计反馈信息子系统、财经信息子系统、贸易信息子系统、交通信息子系统、能源信息子系统、文化教育信息子系统、工业信息子系统、农业信息子系统等。

## (4) 按物理部件组成的结构分类

一个以计算机为基础的信息系统主要由下述部分组成。

① 计算机硬件: 包括中央处理机、输入/输出设备、数据存储设备、数据采集设备及网络通信设备等。

② 计算机软件: 包括系统软件、通用应用软件和专用软件等。

③ 数据库: 存储于物理介质中、经过整理的、可方便进行存取使用和维护的共享数据。数据库的种类很多, 如 FoxPro、Oracle、Sybase 等。

④ 系统规程: 指保证信息系统能正常运行的各种规则、协定、指导手册、说明书等。

⑤ 人: 包括系统管理人员、研制人员、运行维护人员及信息系统的用户。信息系



统能辅助人进行各项工作，并大大减轻人的劳动强度。同时，信息系统又与人密不可分，人的素质直接影响信息系统的效率发挥。

## 2. 信息系统结构的综合

从不同的侧面，可对信息系统进行不同的分解。在信息系统研制的过程中，最常见的方法是将信息系统按职能划分成一个个职能子系统，然后逐个研制和开发。显然，即使每个子系统的性能均很好，也并不能确保整个系统的优良性能，切不可忽略对整个系统的全盘考虑，尤其是各个子系统之间的相互关系应作充分的考虑。因此，在信息系统开发中，强调各子系统之间的协调一致性和整体性。要达到这个目的，就必须在构造信息系统时注意对各种子系统进行统一规划，并对各子系统进行综合。

### （1）横向综合

横向综合，即将同一管理层次的各种职能综合在一起。例如，运行控制层的人事和工资子系统综合在一起，使基层业务处理一体化。

### （2）纵向综合

纵向综合，是把某种职能的各个管理层次的业务组织在一起。这种综合沟通了上下级之间的联系，如工厂的会计系统和公司的会计系统综合在一起，它们有共同之处，能形成一体化的处理过程。

### （3）纵横综合

纵横综合主要是从信息模型和处理模型两个方面来进行综合，做到信息集中共享，程序尽量模块化，注意提取通用部分，建立系统公用数据库和统一的信息处理系统。

## 1.3.5 信息系统的价值

当一个信息系统项目的开发阶段结束，通过测试和必要的鉴定以后，就获得了一个信息系统产品。信息系统作为商品，满足商品的二重性：其使用价值是指信息系统对人们的有用性，即它能满足用户对于信息管理的需要；其价值是指凝结在商品中的人类劳动。无论信息系统是否作为商品进行交换，它的使用价值和价值总是存在的，并且必定同时存在，虽然信息系统使用价值的大小可能会因某种环境因素的变化而受到不同程度的影响。

信息系统作为商品，不仅具有一般物质商品和知识性商品的性质，还具有信息系统自身的特殊性质，表现为以下六个方面。

（1）信息系统中凝结着更多的脑力劳动创造的价值。

（2）信息系统价值实现的间接性。



- (3) 信息系统使用价值与用户水平有直接关系。
- (4) 信息系统软件的易复制性带来的影响。
- (5) 信息系统在使用过程中需要作大量的维护。
- (6) 信息系统价值确定的困难。

信息系统的建设需要一定的投资,对于投资的回报就是信息系统的价值。如何评价一个信息系统的价值,是一个复杂的问题。

以一个公司为例,其信息系统可能有几种不同的价值。

从财务上评价信息系统的价值,是围绕投资回报的问题来考虑的,通常可归结为信息系统产生的效益是否能抵上开发信息系统的成本。评价时有许多财务模型可以使用。

从企业经营战略上考虑,信息系统也有其价值。比如,增强企业的竞争能力,提高对顾客的服务水平等。这些价值是难以定量计算的,但往往比财务方面的价值更重要,有时甚至直接影响着企业的生存。

与节约成本、增加产量等有形价值相比,无形价值是信息系统具有的另一种价值,且信息系统具有更多的无形价值,如计划准确性提高、应变能力增强、反应速度加快等。

价值的多样性和多视角性,需要用多种不同的模型和方法来评估。如果考虑某些价值能否实现的不确定性,则还必须在评估中考虑风险因素。这一切使得信息系统价值的评估变得复杂而困难,需要综合运用多种方法,才能做出较正确的投资决策。

### 1.3.6 信息系统的评价

信息系统的评价是在信息系统的各个阶段,采用科学的评价方法从整体上对信息系统做出正确的评估。其目的是保证设计和开发的信息系统能满足实际要求,并随着信息系统的发展而进一步深化,最终能发挥其最大作用。信息系统的评价本身是一件复杂的工作,在信息系统的各个开发阶段都需要对阶段成果进行评价。由于涉及的内容广泛,实际情况各不相同,因此,没有一个统一的信息系统的评价体系。通常,将信息系统的评价分为两个阶段,即开发设计阶段和运行阶段。

开发设计阶段的评价是指对开发过程的每一个步骤形成的阶段成果进行评价,预测各个阶段对最终信息系统的影响,消除存在的问题,努力确保最终信息系统能正确反映系统需求。由于这一阶段的评价是预测性的,同时,评价结果受到许多不确定因素的影响,所以评价结果未必一定正确。在这一阶段,评价方法因信息系统的分析、设计和开发方法的不同而各不相同。

运行阶段的评价是在系统投入运行以后,由于系统资源和环境的变化,需要对信息系统的运行状况和效益及时进行分析评价,并以此作为系统维护、更新或进一步推

广的依据。这一阶段的评价内容主要包括系统质量评价和系统效益评价。

### （1）系统质量评价

所谓质量评价，就是在一定的范围和一定的条件下对某一事物的优劣程度进行鉴定。评价的关键是选定评价质量指标和评定优劣标准。质量和优劣都是相对的概念，是在一定的条件下相对满意的标准，实际上，不同的人对同一事物的评价甚至可能得出完全相反的结论，因此评价的标准和指标应是被大多数人认可，并在实践中检验是行之有效的，事实上，永远不可能存在绝对意义上的最优。可以选择以下一些评价的参考指标和标准。

① 有效性。考察信息系统的整体功能是否达到了预期的目的，对所要解决的问题是否有效，系统是否能充分反映用户需求等。

② 实用性。考察系统对组织工作有怎样的作用，是否确实可用，用户的满意程度如何等。

③ 可靠性。考察系统运行是否稳定可靠，系统的容错能力和故障恢复的手段是否完备，有无存在重大的问题和隐患亟待改进或解决等。

④ 灵活性。考察系统的兼容能力大小，有无可伸缩性和可扩充性，适应环境变化的灵敏性如何等。

⑤ 安全性。考察系统的安全保密性能如何，有无安全保护措施及系统的安全级别等。

⑥ 易用性。考察用户利用信息系统的方便程度，用户对信息系统接受的难易程度，信息系统对使用人员知识水平的要求程度等。

⑦ 信息质量。考察系统提供信息的响应速度、精确程度等，包括输入、输出信息的要求、质量，对操作人员、管理人员和决策者的作用和价值等。

### （2）系统效益评价

信息系统评价的另一面是系统效益评价。它主要衡量系统对用户的影响，即信息系统的开发和运行给用户带来了多大的收益。从总体上评价信息系统效益是十分困难的，因为信息系统的效益具有整体综合性、形式多样性和时间滞后性等特点，既有直接效益，又有间接效益，既可表现为经济效益，又可表现为社会效益。例如，组织工作效率、劳动生产率的提高，对各种资源的利用率的提高，产品产量、质量的提高，成本降低等，都是直接效益；而影响组织的发展战略、改善管理机制、优化管理效果等都可视为是间接效益。

由于间接效益很难评价和衡量，因此效益评价一般是对其直接效益进行评价的，主要通过直接经济效益来衡量。

## 1.4 信息系统工程

信息系统工程是20世纪80年代出现的以建立信息系统为目标的新兴学科，主要研究各级各类信息系统建设和管理中的规律性的问题。

### 1.4.1 基本概念

信息系统工程是用系统工程的原理、方法来指导信息系统建设与管理的一门工程技术学科。作为系统工程的一个分支，信息系统工程处在发展中，至今还没有统一的定义。

信息系统工程具有系统工程的共同特点，其中，最基本的特点是研究方法的整体性、技术应用上的综合性和管理上的科学化。

#### 1. 整体性

研究方法的整体性就是应用系统学中关于整体大于部分之和的思想，不仅把研究对象看作一个整体，而且把研究过程也看作一个整体。把系统看作是由若干个子系统有机结合的整体来分析与设计。对各子系统的技术要求首先是从实现整个系统技术协调的观点来考虑，从总体协调的需求来制定方案。此外，还要求把所研究的系统放在更大的系统空间或系统环境中去，作为从属于更大系统的组成部分来考虑。对它的所有技术要求，都尽可能从实现与这个更大系统技术协调或适应系统环境的观点来考虑。

#### 2. 综合性

技术应用上的综合性就是系统学中的最优化原则，综合应用各种学科和技术领域内所取得的成就，构筑合理的技术结构，使各种技术相互配合而达到系统整体的最优化。对信息系统而言，它是信息科学、系统科学、管理科学、计算机科学、控制理论及通信科学等各领域技术的综合体。对技术的使用来说，并非每个子系统或部件都要有最好的性能才能获得系统的最佳性能。只要技术结构合理，用廉价的一般部件也可能组合出系统的最佳性能。综合不是各种技术的堆砌，而是以最优化为原则，注重各种技术的协调和结构合理。

#### 3. 科学化

管理上的科学化就是对工程进行科学管理。一个复杂的信息系统工程客观上总存在两个并行进程，一个是工程技术进程，另一个是对工程技术进程的管理控制进程。

后者包括工程的规划、组织、控制、进度安排，对各种方案进行分析、比较和决策、评价选定方案的技术效果等。这些内容称为工程管理。管理的科学化是系统工程的关键。在研制信息系统时，我们强调依照系统工程的方法，明确划分各个工作阶段，保证每个阶段的工作都得到有效的控制管理，对各阶段的工作成果有明确的审查评价标准，最终实现系统的目标。

信息系统工程是信息系统研制和应用的科学方法。有时并不强调方法论的意义，而用工程这一术语作为信息系统建立所进行的一系列活动的总称。这时，信息系统工程也就是指信息系统的整个建立工作。不管是强调方法论，还是强调工程实践活动，其核心问题都是如何进行系统的分析、设计和实现。

应当指出，目前的信息系统工程中很重要的一部分是软件的研制。但就基于计算机的信息系统而言，一方面，系统的最终目标不只是提交软件，而是满足用户要求的使用；另一方面，信息系统的技术构成和研制过程也不只是软件及软件工程，而且还包括硬件、管理及其研制、控制过程等。一般说来，从概念和实践两方面，信息系统工程都有别于软件工程，它包括硬件工程、软件工程及管理工程。

### 1.4.2 信息系统工程的研究方法

信息系统工程的研究涉及多个学科领域，所涉及的主要学科如图 1.16 所示。

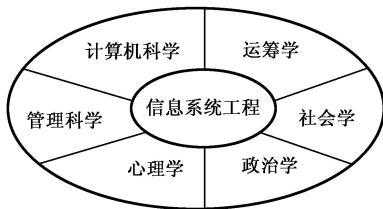


图 1.16 研究信息系统的现代方法

由于信息系统是一个社会技术系统，因此信息系统工程的研究方法不能仅限于工程技术方法。目前，信息系统工程的研究方法分为技术方法、行为方法和社会技术系统方法。

#### 1. 技术方法

信息系统的技术方法重视研究信息系统的规范的数学模型，并侧重于系统的基础理论和技术手段。支持技术方法的学科有计算机科学、数理逻辑、管理科学和运筹学。计算机科学涉及计算理论、计算方法和高效的数据存储和访问方法。数理逻辑侧重于

运用集合论、关系理论等研究信息系统的规范化方法。管理科学着重于管理方法和决策过程的模型的建立。运筹学则强调优化组织的已选参数（如运输、库存控制和交易成本）的数学方法。

## 2. 行为方法

信息系统领域中成长的部分是关于行为问题的。许多行为问题，如系统的使用程度、实施和创造性设计，不能够用技术方法中采用的规范的模型表达，其他行为学科也起着作用。社会学家重视信息系统对群体、组织和社会的作用。政治科学研究信息系统的政治影响和用途。心理学家关注个人对信息系统的反应和人类推理的认知模型。

行为方法不忽视技术。实际上，信息系统技术经常是引发行为问题的因素。但是行为方法的重点一般不在技术方案上，而侧重在态度、管理和组织政策、行为方面。

## 3. 社会技术系统方法

从数据处理系统到管理信息系统再到决策支持系统，这一发展历史告诉我们，信息系统的开发是把计算机科学、数学、管理科学和运筹学的理论研究工作和应用的实践结合起来，并注重社会学、心理学的理论与实践成果。因此，从单一的视角（如技术方法或行为方法）不能有效地把握信息系统的实质。

研究信息系统工程的人员应该了解信息系统涉及的所有学科的观点和看法。事实上，信息系统工程的挑战性和刺激性恰恰使它需要理解和包容许多不同的看法。

社会技术系统方法有助于避免对信息系统采取单纯的技术看法。比如，要正视这一现实：采用信息技术使得成本快速下降和能力迅速增强并不一定能够或不会容易地转化为生产率或利润的提高。

### 1.4.3 信息系统工程的研究范围

一般认为，信息系统工程的目标是，为以计算机和其他信息技术为手段的各类信息系统提供科学的方法、管理手段及有关的工具、标准、规范，通常不包括通信工程、信号处理等具体学科领域的技术。信息系统工程的研究范围如下。

（1）信息系统的基本理论：信息系统的基本观点、认识论和方法论等。

（2）信息系统建模：信息系统概念模型、逻辑模型和物理模型的描述、观察、试验与验证等。

（3）信息系统开发：信息系统建设与管理的概念、方法、评价、规划、工具、标准等一系列相关技术问题和工程问题；依据信息系统工程自身发展的规律和特点，发

展和研究实现信息化建设的工程方法。

（4）信息系统支撑技术在信息系统中的应用：数据库/数据仓库、网络通信、人机交互、分布计算、决策方法、人工智能等技术如何满足信息系统各层次用户的需求，实现业务管理、信息共享、决策分析等功能，并在组织和人的参与下最终达到信息系统的目标。

（5）信息系统集成：研究信息系统集成的原则、方法、技术、工具和有关的标准、规范，应用先进的相关技术，将支持各个信息“孤岛”的小运行环境，集成统一在一个大运行环境中，最终形成一体化的信息系统。

## 习 题

1. 如何认识系统学的一些基本原理对信息系统开发的指导意义？
2. 如何理解信息运动模型与信息系统功能之间的关系？
3. 以一个具体的信息系统为例，说明其四种结构的具体含义。
4. 从管理科学的角度来看，数据、信息、知识三者之间有什么区别？

## 第 2 章 信息系统的基础理论

事实证明，任何技术的发展都离不开理论的支持。目前，信息系统已在各个领域得到了广泛应用，支持其发展的基础理论也日益引起人们的关注。为了能够使大家更好地理解信息系统的概念，掌握信息处理的原理，本章将从数学模型的角度对信息系统的基本理论进行介绍。

本章内容主要供本专业学生和相关领域研究人员学习和参考，一般应用开发技术人员可以跳过本章内容，直接进入后续章节的学习。

### 2.1 集合论基础

集合论中的概念在信息系统理论中得到了广泛的应用，下面将简要介绍一下集合、笛卡儿乘积、关系、函数、划分等有关的概念。

#### 2.1.1 集合及其表示

所谓集合，即是由某些可以相互区分的任意对象汇集在一起所组成的一个整体。组成一个集合的各个对象，称为这个集合的元素。通常，用大写拉丁字母  $A, B, C, \dots$  表示集合，用小写拉丁字母  $a, b, c, \dots$  表示集合的元素。

有限个元素  $x_1, x_2, \dots, x_n$  组成的集合，称为有限集合。无限个元素组成的集合，称为无限集合。例如，整数组成的集合是一个无限集合。我们把不含元素的集合，称为空集，记为  $\Phi$ 。

有限集合  $A$  中不同元素的个数称为集合的基数，表示为  $\#A$  或  $|A|$ 。

设  $a$  为任意一个对象， $A$  为任意一个集合，则在  $a$  和  $A$  之间有且仅有以下两种情况之一出现：

- (1)  $a$  为  $A$  的元素，记为“ $a \in A$ ”或“ $A \ni a$ ”，并称之为“ $a$  属于  $A$ ”或“ $A$  含有  $a$ ”。
- (2)  $a$  不为  $A$  的元素，记为“ $a \notin A$ ”或“ $A \nsubseteq a$ ”，有时也可记为“ $a \bar{\in} A$ ”或“ $A \bar{\ni} a$ ”，并称之为“ $a$  不属于  $A$ ”或“ $A$  不含有  $a$ ”。

设  $A, B$  为任意两个集合，若对每个  $a \in A$  皆有  $a \in B$ ，则称  $A$  为  $B$  的子集或  $B$  包含  $A$ ，并称  $B$  为  $A$  的母集，记为  $A \subseteq B$  或  $B \supseteq A$ 。

若  $A, B$  包含的元素完全相同，则称集合  $A$  和  $B$  相等，记为  $A=B$ 。由定义可知，

$A=B$  的充要条件是  $A \subseteq B$  且  $B \supseteq A$ 。应该指出的是, 一个集合中元素排列的顺序是无关紧要的。

若  $A \subseteq B$  且  $A \neq B$ , 则称  $A$  为  $B$  的真子集或  $B$  真包含  $A$ , 记为  $A \subset B$  或  $B \supset A$ 。

如果所研究的集合皆为某个集合的子集, 则称该集合为全集, 记为  $E$ 。空集  $\Phi$  是任何集合的一个子集。

设  $A$  是一个集合, 则  $A$  的所有子集组成的集合称为  $A$  的幂集, 记为  $2^A$ 。

当  $A$  为有限集时,  $|A| = n$ , 则  $2^A$  的元素个数为

$$C_n^0 + C_n^1 + \cdots + C_n^n = 2^n$$

### 2.1.2 集合的运算与笛卡儿乘积

下面我们介绍集合的运算定义, 设  $A$ 、 $B$  为任意两个集合, 则

$$A \cup B = \{x \mid x \in A \text{ 或 } x \in B\}$$

$$A \cap B = \{x \mid x \in A \text{ 且 } x \in B\}$$

$$A - B = \{x \mid x \in A \text{ 且 } x \notin B\}$$

$$A \oplus B = (A \cup B) - (A \cap B)$$

对于全集  $E$  和集合  $A$ , 称  $E - A$  是集合  $A$  的补集, 记为  $\bar{A}$ 。

对任意集合  $A$ 、 $B$ 、 $C$ , 有如下几条运算律:

- (1)  $A \cup A = A, A \cap A = A$ ;
- (2)  $A \cup B = B \cup A, A \cap B = B \cap A$ ;
- (3)  $(A \cup B) \cup C = A \cup (B \cup C)$ ;  
 $(A \cap B) \cap C = A \cap (B \cap C)$ ;
- (4)  $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ ;  
 $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ ;
- (5)  $A \cup (A \cap B) = A, A \cap (A \cup B) = A$ ;
- (6)  $A \cup \bar{A} = E, A \cap \bar{A} = \Phi$ ;
- (7)  $\overline{A \cup B} = \bar{A} \cap \bar{B}, \overline{A \cap B} = \bar{A} \cup \bar{B}$ ;
- (8)  $E \cup A = E, E \cap A = A$ ;
- (9)  $A \cup \Phi = A, A \cap \Phi = \Phi$ 。

设  $I_+$  为正整数的集合, 则有以下定义。

设  $n \in I_+$  且  $x_1, x_2, \dots, x_n$  为  $n$  个任意的元素:

- (i) 若  $n=1$ , 则令  $\langle x_1 \rangle = \{x_1\}$ ;
- (ii) 若  $n=2$ , 则令  $\langle x_1, x_2 \rangle = \{\{x_1\}, \{x_1, x_2\}\}$ ;



(iii) 若  $n > 2$ , 则令  $\langle x_1, x_2, \dots, x_n \rangle = \langle \langle x_1, x_2, \dots, x_{n-1} \rangle, x_n \rangle$ 。

称  $\langle x_1, x_2, \dots, x_n \rangle$  为由  $x_1, x_2, \dots, x_n$  组成的  $n$  元序偶, 并称每个  $x_i$  ( $1 \leq i \leq n$ ) 为它的第  $i$  个分量。序偶的元素排列是有顺序的, 不能任意颠倒, 因此两个序偶相等, 应该是对应元素相等。

设  $n \in I_+$  且  $A_1, A_2, \dots, A_n$  为  $n$  个任意集合, 若令

$$A_1 \times A_2 \times \dots \times A_n = \{ \langle x_1, x_2, \dots, x_n \rangle \mid \text{若 } 1 \leq i \leq n, \text{ 则 } x_i \in A_i \}$$

则称  $A_1 \times A_2 \times \dots \times A_n$  为  $A_1, A_2, \dots, A_n$  的笛卡儿乘积, 简记为  $\prod_{i=1}^n A_i$ , 并称  $n$  为  $A_1 \times A_2 \times \dots \times A_n$  的维数。当  $A_1 = A_2 = \dots = A_n = A$  时, 可把  $A_1 \times A_2 \times \dots \times A_n$  简记为  $A^n$ 。

### 2.1.3 关系

关系的概念在数学中是常用的, 诸如大于、小于、等于、包含等都属于关系。下面给出关系的定义。

设  $A$  是一个集合,  $A \times A$  的一个子集  $R$ , 称为是集合  $A$  上的一个二元关系, 简称关系。

对于  $a \in A, b \in A$ , 如果  $\langle a, b \rangle \in R$ , 则称  $a$  和  $b$  存在关系  $R$ , 记为  $aRb$ ; 如果  $\langle a, b \rangle \notin R$ , 则称  $a$  和  $b$  不存在关系  $R$ , 记为  $a \not R b$ 。

设有集合  $A$ ,  $R$  是  $A$  上的关系:

- (1) 对每个  $a \in A$ , 如果有  $aRa$ , 则称  $R$  是自反的;
- (2) 对于  $a, b \in A$ , 如果有  $aRb$ , 又有  $bRa$ , 则称  $R$  是对称的;
- (3) 对于  $a, b, c \in A$ , 如果有  $aRb$  和  $bRc$ , 则有  $aRc$ , 此时称  $R$  是传递的;
- (4) 对每个  $a \in A$ , 如果有  $aRa$ , 则称  $R$  是反自反的。

例如, 数之间的相等关系, 具有自反性、对称性和传递性, 小于关系和大于关系没有自反性, 但有传递性。

设  $R$  是非空集合  $A$  上的一个关系, 如果  $R$  有自反性、对称性和传递性, 则称  $R$  是一个等价关系。

由等价关系  $R$  可以把  $A$  分为若干个子集, 每个子集称为一个等价类, 同一等价类中的元素是互相等价的。

**【例 2.1】** 设  $Z$  是整数集合,  $R$  是  $Z$  上模 3 同余的关系, 即

$$R = \{ \langle x, y \rangle \mid x, y \in Z, \text{ 且 } x \equiv y \pmod{3} \}$$

由于  $R$  是等价关系, 所以存在三个等价类:

$$[0]_R = \{ \dots, -6, -3, 0, 3, 6, \dots \}$$

$$[1]_R = \{ \dots, -5, -2, 1, 4, 7, \dots \}$$

$$[2]_R = \{\dots, -4, -1, 2, 5, 8, \dots\}$$

其中  $[0]_R$ ,  $[1]_R$ ,  $[2]_R$  是表示等价类的符号。

设  $R$  是集合  $A$  上的一个关系, 则

$$R^{-1} = \{\langle y, x \rangle \mid x, y \in A \text{ 且有 } \langle x, y \rangle \in R\}$$

则称  $R^{-1}$  是关系  $R$  的逆关系。

例如, 小于关系的逆关系是大于关系, 相等关系的逆关系仍然是相等关系。

#### 2.1.4 关系的闭包

设  $R$  是集合  $A$  上的关系, 如果另有关系  $R'$  满足:

(1)  $R'$  是传递的 (自反的, 对称的);

(2)  $R' \supseteq R$ ;

(3) 对任何传递的 (自反的, 对称的) 关系  $R''$ , 当有  $R'' \supseteq R$  时, 就有  $R'' \supseteq R'$ 。

则称关系  $R'$  是  $R$  的传递 (自反的, 对称的) 闭包。

$R$  的传递闭包表示为  $t(R)$ ,  $R$  的自反闭包表示为  $r(R)$ ,  $R$  的对称闭包表示为  $s(R)$ 。

如果给定一个集合  $A$  上的关系  $R$ , 可用以下方法找到传递闭包  $t(R)$ 、自反闭包  $r(R)$  和对称闭包  $s(R)$ :

(1)  $t(R) = R \cup R^2 \cup \dots \cup R^n$ , 其  $|A| = n$ ;

(2)  $r(R) = R \cup I_A$ , 其中  $I_A = \{\langle x, x \rangle \mid x \in A\}$ ;

(3)  $s(R) = R \cup R^{-1}$ 。

#### 2.1.5 函数映射

映射是关系的一个特殊类型, 还可被称为函数。

设集合  $X$  和  $Y$ ,  $f$  是从  $X$  到  $Y$  的一个关系, 如果对每一个  $x \in X$ , 有唯一的  $y \in Y$ , 使得  $\langle x, y \rangle \in f$ , 则称关系  $f$  是函数, 记为  $f: X \rightarrow Y$ 。

如果存在  $\langle x, y \rangle \in f$ , 则  $x$  是  $f$  的自变量,  $y$  是  $f$  作用下的像点, 可记为  $y = f(x)$ 。

函数  $f$  的定义域  $\text{dom } f$  定义为:

$$\text{dom } f = \{x \mid x \in X \text{ 且有 } y \in Y \text{ 使 } y = f(x)\}$$

由定义可知  $\text{dom } f = X$ 。

函数  $f$  的值域  $\text{ran } f$  定义为

$$\text{ran } f = \{y \mid y \in Y \text{ 且有 } x \in X \text{ 使 } f(x) = y\}$$

显然  $\text{ran } f \subseteq Y$ 。

函数有以下几个特殊类型。

(1) 对于  $f: X \rightarrow Y$ , 如果  $f$  的值域  $\text{ran } f = Y$ , 即  $Y$  的每一个元素都是  $X$  中一个或多个元素的像点, 则称  $f$  是满射的。

(2) 对于  $f: X \rightarrow Y$ , 如果  $X$  中没有两个元素有相同的像点, 即对于任意  $x_1, x_2 \in X$ :

如果  $x_1 \neq x_2$ , 则有  $f(x_1) \neq f(x_2)$ ,

或者,

如果  $f(x_1) = f(x_2)$ , 则有  $x_1 = x_2$ 。

则称  $f$  是入射的。

(3) 对于  $f: X \rightarrow Y$ , 如果  $f$  既是满射的, 又是入射的, 则称  $f$  是双射的, 或称  $f$  是一一对应的。

设  $f: X \rightarrow Y, g: Y \rightarrow Z$ , 则由  $f, g$  确定的  $X$  到  $Z$  的函数  $h: x \mapsto g(f(x))$   $\forall x \in X$ , 叫做函数  $f, g$  的合成, 记为  $h = g \circ f$ 。

### 2.1.6 集合的划分

设非空集合  $A, \Pi = \{\pi_1, \pi_2, \dots, \pi_n\}$ , 其中  $\pi_i \subseteq A, \pi_i \neq \Phi (i=1, 2, \dots, n)$ ,

如果有  $\bigcup_{i=1}^n \pi_i = A$  且  $\pi_i \cap \pi_j = \Phi (i \neq j)$ , 则称  $\Pi$  是  $A$  的划分, 其中  $\pi_i$  是一个划分块。

**【例 2.2】** 集合  $S = \{a, b, c, d\}$ , 考虑下列集合:

$$A = \{\{a, b\}, \{c, d\}\}$$

$$B = \{\{a\}, \{b\}, \{c\}, \{d\}\}$$

$$C = \{\{a\}, \{b, c, d\}\}$$

$$D = \{\{a, b, c, d\}\}$$

$$E = \{\{a, b\}, \{b, c, d\}\}$$

$$F = \{\{a, b\}, \{c\}\}$$

则  $A, B, C$  和  $D$  都是  $S$  的划分, 而  $E$  和  $F$  不是  $S$  的划分。

设  $R$  为非空集合  $A$  上的一个等价关系, 由  $R$  把  $A$  划分成为若干子集, 即等价类, 这些等价类组成集合  $A$  的一个划分。由不同的等价关系可以导出集合的不同划分。

## 2.2 信息系统的数学模型

信息系统就是对输入的原始数据进行加工处理而产生信息输出的系统。从集合论的观点来看, 信息系统可以被看作是一些对象的有限集合  $O$ , 即

$$O = \{o_1, o_2, \dots, o_n\} \subseteq U$$

其中,  $o_i \in U$  称为对象,  $U$  称为对象的论域。

属性是对象的表现形式，所有的对象都可以用它们属性的有限集合  $A$  进行描述，即

$$A = \{a_1, a_2, \dots, a_m\}$$

$A$  中的每个属性  $a$  的定义域为  $V_a$ ，即每个属性  $a$  与一个非空的  $a$  的值的集合  $V_a$  相联系。

为了确定对象的属性值，定义一个函数  $\rho: O \times A \rightarrow V$ ，其中

$$V = \bigcup_{a \in A} V_a$$

对象  $o_i$  的属性  $a_j$  的值用  $\rho(o_i, a_j)$  表示。

有了以上的这些说明，下面给出信息系统的数学模型。

信息系统是如下一个集合结构：

$$S = \langle O, A, V, \rho \rangle$$

即信息系统  $S$  是一个四元组，其中

$O$ —对象的有限集合，用  $o_i$  表示  $O$  中的具体对象。

$A$ —对象属性的有限集合，用于区别和分类每一个对象。元素  $a_i$  表示  $A$  中的具体属性。

$V$ —每个属性定义域  $V_a$  的和集，即  $V = \bigcup_{a \in A} V_a$ 。

$\rho$ —对象属性值函数，是从  $O \times A$  到  $V$  的映射，用  $\rho(o_i, a_j)$  表示对象  $o_i$  的属性  $a_j$  的值。

下面举例说明信息系统的结构。

**【例 2.3】** 假设有一学生管理信息系统  $S = \langle O, A, V, \rho \rangle$ ，其中

对象集  $O = \{\text{学生 1, 学生 2, } \dots, \text{学生 100}\}$

属性集  $A = \{\text{性别, 年龄, 年级, 成绩}\}$

定义域和集  $V = \{V_{\text{性别}} \cup V_{\text{年龄}} \cup V_{\text{年级}} \cup V_{\text{成绩}}\}$

其中， $V_{\text{性别}} = \{\text{男, 女}\}$ ， $V_{\text{年龄}} = \{x \mid 16 \leq x \leq 25, x \text{ 是整数}\}$ ， $V_{\text{年级}} = \{\text{大一, 大二, 大三, 大四}\}$ ， $V_{\text{成绩}} = \{\text{优, 良, 中, 及格, 差}\}$ 。

函数  $\rho$  可用表 2.1 定义。

表 2.1 函数  $\rho$  的定义

对象	性别	年龄	年级	成绩
学生 1	男	20	大二	良
学生 2	女	19	大二	优
...	...	...	...	...
学生 99	男	21	大三	良
学生 100	男	20	大三	中

## 2.3 信息系统的性质

### 2.3.1 对象的信息

我们称二元组  $(a, v)$  (其中  $a \in A, v \in V_a$ ) 为属性  $a$  的描述。在例 1.3 中, (性别, 女)、(年龄, 19)、(年级, 大二)、(成绩, 优) 等都是相应属性的描述。

在 2.2 节里, 定义了对象属性值函数  $\rho$ , 它是从  $O \times A \rightarrow V$  的映射, 对象  $o_i$  的属性  $a_j$  的值用  $\rho(o_i, a_j)$  来表示。现在对任意的  $o \in O$ , 定义由  $A$  到  $V$  的函数  $\rho_o, \rho_o(a) = \rho(o, a)$ 。这里, 把函数  $\rho_o$  称为信息系统  $S$  关于对象  $o$  的信息。

所谓信息系统的信息量, 就是该信息系统中不同的信息的个数。

一个信息系统中的所有对象, 可以用它们的属性的有限集合  $A$  加以描述, 而  $A$  中的每个属性  $a$  的定义域是  $V_a$ , 将属性  $a$  所取的值的个数用  $\text{card}(V_a)$  表示, 则一个信息系统  $S$  中的信息量共有  $\prod_{a \in A} \text{card}(V_a)$  个。

同例 2.3, 该学生管理信息系统中有 400 个不同的信息:

$$\begin{aligned} & \text{card}(V_{a1}) \cdot \text{card}(V_{a2}) \cdot \text{card}(V_{a3}) \cdot \text{card}(V_{a4}) \\ &= \text{card}(V_{\text{性别}}) \cdot \text{card}(V_{\text{年龄}}) \cdot \text{card}(V_{\text{年级}}) \cdot \text{card}(V_{\text{成绩}}) \\ &= 2 \cdot 10 \cdot 4 \cdot 5 \\ &= 400 \end{aligned}$$

设一个信息系统  $S$  中所有信息的总和为  $\text{INF}(S)$ , 那么, 对于每一个信息  $\rho' \in \text{INF}(S)$ , 定义

$$\begin{aligned} O_{\rho'} &= \{o \in O \mid \rho_o = \rho'\} \\ &= \{o \in O \mid \bigcap_{a \in A} \rho_o(a) = \rho'(a)\} \end{aligned}$$

则  $O_{\rho'}$  是  $O$  中信息为  $\rho'$  的所有“对象”的集合。

$$O_{\rho'} = \bigcap_{a \in A} \{o \in O \mid \rho_o(a) = \rho'(a)\}$$

从这个定义可以知道, 在  $O$  中信息为  $\rho'$  的这些“对象”, 在信息系统中它们的属性都是相同的, 这说明用对象的属性是无法区分这类“对象”的。

如果  $O_{\rho'} = \Phi$ , 则称信息  $\rho'$  为空信息, 否则称为非空信息。

**【例 2.4】** 设有一信息系统  $S = \langle O, A, V, \rho \rangle$ , 其函数  $\rho$  用表 2.2 定义。

表 2.2 定义函数  $\rho$ 

$O$	$A_1$	$a_2$	$a_3$	$O$	$a_1$	$a_2$	$a_3$
$O_1$	$E_1$	$f_2$	$r_1$	$o_3$	$e_1$	$f_2$	$r_1$
$O_2$	$E_2$	$f_3$	$r_2$	$o_4$	$e_1$	$f_3$	$r_3$

设  $\rho'$  是信息系统  $S$  中的一个信息,  $\rho'(a_1) = e_1, \rho'(a_2) = f_2, \rho'(a_3) = r_1$ , 下面计算  $O_{\rho'}$ :

$$\begin{aligned}
 O_{\rho'} &= \{o \in O \mid \rho_o = \rho'\} \\
 &= \bigcap_{a \in A} \{o \in O \mid \rho_o(a) = \rho'(a)\} \\
 &= \{o \in O \mid \rho(o, a_1) = e_1\} \cap \{o \in O \mid \rho(o, a_2) = f_2\} \cap \{o \in O \mid \rho(o, a_3) = r_1\} \\
 &= \{o_1, o_3, o_4\} \cap \{o_1, o_3\} \cap \{o_1, o_3\} \\
 &= \{o_1, o_3\}
 \end{aligned}$$

### 2.3.2 最大的信息系统

建立信息系统就是为了对现实世界中的信息进行描述。为了尽可能地反映完整的现实世界, 就应该使信息系统中存储的信息尽量完备。

如果一个信息系统  $S$  中的每一个信息都是非空的, 即对任意的  $\rho' \in \text{INF}(S)$ , 有  $O_{\rho'} \neq \Phi$ , 则称信息系统  $S$  为最大系统或完全系统。

同例 2.4: 现在看看这个信息系统是不是最大的?

设  $\rho'$  是信息系统  $S$  中的一个信息,  $\rho'(a_1) = e_1, \rho'(a_2) = f_3, \rho'(a_3) = r_3$ , 则

$$\begin{aligned}
 O_{\rho'} &= \{o \in O \mid \rho_o = \rho'\} \\
 &= \{o \in O \mid \rho(o, a_1) = e_1\} \cap \{o \in O \mid \rho(o, a_2) = f_3\} \cap \{o \in O \mid \rho(o, a_3) = r_3\} \\
 &= \{o_1, o_3, o_4\} \cap \{o_2\} \cap \{o_4\} \\
 &= \Phi
 \end{aligned}$$

这说明该信息系统  $S$  中存在空信息, 这个空信息就是  $\rho', \rho'(a_1) = e_1, \rho'(a_2) = f_3, \rho'(a_3) = r_3$ , 因此该信息系统不是最大的。

对于某一特定的信息系统, 其相应的最大系统不一定存在, 即对于该系统的属性集合, 属性值的任意一个组合不一定均有一个实体对象与之对应。因此, 最大的信息系统是一个理想化的概念。引入这个概念就是为了以其为标准, 希望能够将信息系统尽量完备化。当然在现实世界中, 某一领域的最大系统也是存在的, 也就是说, 对于某些信息系统, 存在与其对应的最大系统。

那么如何使一个已有的信息系统  $S$  尽量完备化呢?

首先, 需要判断该系统  $S$  是否为最大的信息系统。设  $Q = \{\rho' \in \text{INF}(S) \mid O_{\rho'} = \Phi\}$ ,

由定义可知, 若  $Q=\Phi$ , 则系统  $S$  为最大的信息系统。否则, 设  $O' = \bigcup_{\rho \in Q} \{o \in U \mid \rho_o = \rho'\}$ , 则集合  $O'$  为不属于  $S$  的一个未知对象的集合, 其中包含有可能存在的实体对象, 故对集合  $O'$  中的每一个元素进行考察, 将合理的对象逐一加入系统  $S$  中, 即可将系统  $S$  完备化。此外, 利用集合  $O'$  还可用于对未知对象的预测, 提供对象的范围和属性特征。

### 2.3.3 可选的信息系统

如果在  $O$  中信息为  $\rho'$  的“对象”只有一个, 也就是说, 一个信息  $\rho'$  唯一地描述一个对象  $o$ , 即  $|O_{\rho'}| = 1$ , 则称这样的信息为可选的。

如果一个信息系统  $S$  中的每一个非空信息  $\rho'$  都是可选的, 则称该信息系统为可选的信息系统。

信息系统提供的是真实世界中对象的信息, 但是这些信息可能并不足以精确地描述相应的对象。有时, 有些对象具有相同的属性值。当两个对象的所有属性值相同时, 这两个对象就难以辨识。所以, 以信息系统现有的对象信息可能并不足以唯一确定每个对象。

如果将关系型数据库中的列视为属性、行视为对象的话, 也可以将其看作是一种信息系统。其中,  $p$  行  $r$  列的元素值即可为对象  $o_p$  的属性  $a_r$  的值。关系型数据库的每一行给出了信息系统中相应对象的信息, 而且信息  $\rho'$  与对象  $o$  是一一对应的。所以, 可以认为关系型数据库是可选的信息系统。

同例 2.4: 由 2.3.1 节中的计算可知, 对于一个信息  $\rho'$ ,  $\rho'(a_1) = e_1, \rho'(a_2) = f_2, \rho'(a_3) = r_1, |O_{\rho'}| = 2$ 。这说明这个信息系统不是可选的。

### 2.3.4 信息系统的划分

由可选的定义可知, 当一个信息系统不是可选的时候, 必存在信息  $\rho'$ , 有  $|O_{\rho'}| > 1$ 。将这些属性值相同的对象划为一类, 即可得到信息系统的一个划分, 而所得的各类元素集合将被视为若干个等价类。

基于上述思想, 下面给出相应的定义。

#### 1. 关系对象

所谓关系对象, 就是指具有相同属性的一组对象, 它是一种关系结构。下面确定两个二元关系  $\tilde{a}$  和  $\tilde{S}$ 。

#### (1) 定义关系

$$\tilde{a} = \{\langle x, y \rangle \mid \rho(x, a) = \rho(y, a); x, y \in O; a \in A\}$$

即如果对于属性  $a$ , 对象  $x$  和  $y$  具有相同的属性值, 则对象  $x$  和  $y$  组成的二元关系就属于关系  $\tilde{a}$ , 记为  $x\tilde{a}y$ 。

## (2) 定义关系

$$\tilde{S} = \{\langle x, y \rangle \mid \rho_x = \rho_y; x, y \in O\}$$

即如果对象  $x$  和  $y$  具有相同的信息, 则对象  $x$  和  $y$  组成的二元关系就属于关系  $\tilde{S}$ , 记为  $x\tilde{S}y$ 。对于  $A$  中的任意一个属性  $a$ , 这两个对象都具有相同的属性值。

(3) 对任何一个信息系统  $S = \langle O, A, V, \rho \rangle$ , 关系  $\tilde{a}$ 、 $\tilde{S}$ , 都是等价关系, 且

$$\tilde{S} = \bigcap_{a \in A} \tilde{a}$$

若两个对象对  $A$  中的每一个属性  $a$  都存在着  $\tilde{a}$  关系, 则这两个对象具有相同的信息, 即有  $\tilde{S}$  关系。这说明关系  $\tilde{a}$  和  $\tilde{S}$  是等价关系。

类似的, 对于属性组  $B \subseteq A$ , 可定义关系

$$\tilde{B} = \{\langle x, y \rangle \mid \rho(x, a) = \rho(y, a); x, y \in o; \forall a \in B\} = \bigcap_{a \in B} \tilde{a}$$

即如果对于属性组  $B$  中的所有属性, 对象  $x$  和  $y$  均具有相同的属性值, 则对象  $x$  和  $y$  组成的二元关系就属于关系  $\tilde{B}$ , 记为  $x\tilde{B}y$ 。

如例 2.4: 因为  $\rho_{o_1}(a_1) = \rho_{o_4}(a_1) = e_1$ , 所以  $o_1\tilde{a}o_4$ , 即对象  $o_1$  和  $o_4$  具有相同的属性  $a_1$ , 也就是说只用属性  $a_1$  是无法区分  $o_1$  和  $o_4$  的, 因此  $o_1$  和  $o_4$  是属于关系  $\tilde{a}$  的。

同样, 因为  $\rho_{o_1} = \rho_{o_3} = \{e_1, f_2, r_1\}$ , 所以  $o_1\tilde{S}o_3$ , 即对象  $o_1$  和  $o_3$  对  $A$  中相应属性都有相同的值  $\{e_1, f_2, r_1\}$ , 因此  $o_1$  和  $o_3$  是属于关系  $\tilde{S}$  的。

## 2. 关系 $\tilde{S}$ 的等价类

从关系对象的描述中, 可以得出这样一个结论: 某些对象如果是关系对象, 则可以把它们划分为一个等价类, 这个等价类要么就是具有相同的属性  $a$ , 要么就是具有相同的信息。这样, 等价类是由关系  $\tilde{a}$  或  $\tilde{S}$  来划分的, 所以也可以把关系  $\tilde{a}$  和  $\tilde{S}$  称为划分。

划分  $\tilde{S}$  把对象集  $O$  分成若干个等价类, 而每个等价类具有一个相同的非空信息。这表明一个等价类与一个信息是对应的。

设  $\tilde{K}$  是一个划分, 它把对象集  $O$  分成若干个等价类, 这些等价类的集合, 称之为近似空间, 记为  $\Omega = (O, \tilde{K})$ 。因此, 划分  $\tilde{S}$  也生成了一个近似空间, 记为  $\Omega = (O, \tilde{S})$ 。

同例 2.4: 将信息系统  $S$  按各属性划分。

(1)  $\tilde{a}_1$  的划分由两个等价类组成:  $\{o_1, o_3, o_4\}, \{o_2\}$ 。

(2)  $\tilde{a}_2$  的划分由两个等价类组成:  $\{o_1, o_3\}, \{o_2, o_4\}$ 。

(3)  $\tilde{a}_3$  的划分由三个等价类组成:  $\{o_1, o_3\}, \{o_2\}, \{o_4\}$ 。

(4) 划分  $\tilde{S} = \tilde{a}_1 \cap \tilde{a}_2 \cap \tilde{a}_3$  由三个等价类组成, 即  $\Omega = (O, \tilde{S}) = \{\{o_1, o_3\}, \{o_2\}, \{o_4\}\}$ 。



关系  $\tilde{S}$  的等价类叫做  $S$  的基本集, 因此划分  $\tilde{S} = \tilde{a}_1 \cap \tilde{a}_2 \cap \tilde{a}_3$  也可以说成是由三个基本集组成的。

对一个等价类  $X_i$ , 用  $\text{Des}(X_i) = (a, v)$  来对其特性进行描述, 其中  $a \in A, v \in V_a$ 。在上例中,  $\Omega = (O, \tilde{S}) = (X_1, X_2, X_3)$ , 则等价类  $X_1 = \{o_1, o_3\}$  的描述是:  $\text{Des}(X_1) = (a_1 = e_1, a_2 = f_2, a_3 = r_1)$ 。

在上例中, 按属性值把所有对象  $O$  划分成若干个等价类 (基本集), 每个等价类中的对象是不可区分的。一般地说, 一个等价类 (基本集) 中可能会有一个以上的对象, 即系统是可选的。这表明, 或者是所选取的属性及其定义域, 尚不足以描述和区分  $O$  中的每一个对象; 或者是一个等价类中存在多余的对象, 即存在冗余。在前一种情况下, 可通过联系实际情况追加更多的属性, 以达到描述和区分  $O$  中每一个对象的目的。而在后一种情况下, 可通过进一步的筛选删去多余的对象。

### 3. 等价类之间的最小包含 (下近似) 和最大包含 (上近似)

假设  $B, C \subseteq A, \Omega_1 = (O, \tilde{B}), \Omega_2 = (O, \tilde{C})$ , 对于这两个划分之间的关系, 用下面几个概念进行说明。

(1)  $X_j (X_j \in \Omega_2)$  中最小包含  $\Omega_1$  中的等价类有:

$$\underline{BX}_j = \bigcup \{B_i \mid B_i \in \Omega_1 \text{ 且 } B_i \subseteq X_j\};$$

$\underline{BX}_j$  是  $\Omega_1$  中被  $X_j$  包含的等价类的并集。

(2)  $X_j (X_j \in \Omega_2)$  中最大包含  $\Omega_1$  中的等价类有:

$$\overline{BX}_j = \bigcup \{B_i \mid B_i \in \Omega_1 \text{ 且 } B_i \cap X_j \neq \Phi\};$$

$\overline{BX}$  是  $\Omega_1$  中与  $X_j$  的交集非空的等价类的并集。

(3) 在近似空间  $\Omega_1$  中, 集合  $X_j$  的边界定义为

$$Bn_B(X_j) = \overline{BX}_j - \underline{BX}_j$$

显然,  $Bn_B(X_j)$  中的对象根据  $B$  中各属性的属性值是不能确定它们是否属于等价类  $X_j$  的。

(4) 确定度:

$$\alpha_B(X_j) = \frac{|O| - |Bn_B(X_j)|}{|O|}$$

$\alpha_B(X_j)$  的值反映  $O$  中能够根据  $B$  中各属性的属性值能确定其属于或不属于  $X_j$  的比例, 也即对  $O$  中的任意一个对象, 根据  $B$  中各属性的属性值确定它属于或不属于  $X_j$  的可信度。显然,  $0 \leq \alpha_B(X_j) \leq 1$ 。

当  $\alpha_B(X_j) = 1$  时,  $O$  中的全部对象根据  $B$  中各属性的属性值就可以确定其是否属于  $X_j$ 。

当  $0 < \alpha_B(X_j) < 1$  时,  $O$  中的部分对象根据  $B$  中各属性的属性值可以确定其是否属于  $X_j$ , 而另一部分对象是不能确定其是否属于  $X_j$  的。

当  $\alpha_B(X_j) = 0$  时,  $O$  中的全部对象都不能根据  $B$  中各属性的属性值确定其是否属于  $X_j$ 。

**【例 2.5】** 设有一信息系统  $S = \langle O, A, V, \rho \rangle$ , 其中函数  $\rho$  由表 2.3 定义。

表 2.3 定义函数  $\rho$

$O$	$a_1$	$a_2$	$a_3$	$O$	$a_1$	$a_2$	$a_3$
$o_1$	$e_{11}$	$e_{20}$	$e_{32}$	$o_6$	$e_{12}$	$e_{20}$	$e_{30}$
$o_2$	$e_{10}$	$e_{21}$	$e_{30}$	$o_7$	$e_{10}$	$e_{21}$	$e_{31}$
$o_3$	$e_{12}$	$e_{20}$	$e_{30}$	$o_8$	$e_{11}$	$e_{21}$	$e_{31}$
$o_4$	$e_{11}$	$e_{21}$	$e_{30}$	$o_9$	$e_{11}$	$e_{20}$	$e_{32}$
$o_5$	$e_{11}$	$e_{20}$	$e_{32}$	$o_{10}$	$e_{10}$	$e_{21}$	$e_{31}$

属性组  $C = \{a_1, a_2\}, D = \{a_3\} \subseteq A, \Omega_2 = (O, \tilde{D}) = \{X_1, X_2, X_3\}$ :

$X_1 = \{o_1, o_5, o_9\}, X_2 = \{o_7, o_8, o_{10}\}, X_3 = \{o_2, o_3, o_4, o_6\}$ ;

$\Omega_1 = (O, \tilde{C}) = \{Y_1, Y_2, Y_3, Y_4\}$ :

$Y_1 = \{o_1, o_5, o_9\}, Y_2 = \{o_2, o_7, o_{10}\}, Y_3 = \{o_3, o_6\}, Y_4 = \{o_4, o_8\}$ ;

根据上述定义,  $\underline{C}X_1 = \{o_1, o_5, o_9\}, \underline{C}X_2 = \{\}, \underline{C}X_3 = \{o_3, o_6\}$ 。

因为  $Y_1 \cap X_1 \neq \Phi, Y_2 \cap X_1 = \Phi, Y_3 \cap X_1 = \Phi, Y_4 \cap X_1 = \Phi$ , 所以根据定义得:

$\bar{C}X_1 = \{o_1, o_5, o_9\}$

$\bar{C}X_2 = \{o_2, o_4, o_7, o_8, o_{10}\}$

$\bar{C}X_3 = \{o_2, o_3, o_4, o_6, o_7, o_8, o_{10}\}$

$Bn_C(X) = \bar{C}X - \underline{C}X = \{\{\}, \{o_2, o_4, o_7, o_8, o_{10}\}, \{o_2, o_3, o_6, o_7, o_{10}\}\}$

则

$$\alpha_C(X_1) = \frac{10-0}{10} = 1.0$$

$$\alpha_C(X_2) = \frac{10-5}{10} = 0.5$$

$$\alpha_C(X_3) = \frac{10-5}{10} = 0.5。$$

$S$  中的 10 个对象, 都可以根据  $C$  中各属性的属性值确定其是否属于  $X_1$ ; 50% 的对象可以根据  $C$  中各属性的属性值确定其是否属于  $X_2$ ; 50% 的对象可以根据  $C$  中各属性的属性值确定其是否属于  $X_3$ 。

对于  $X_3$ ,  $\bar{C}X_3$ 、 $\underline{C}X_3$  和  $\alpha_C(X_3)$  可用图 2.1 来说明。

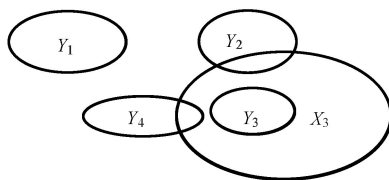


图 2.1 等价类之间的最小包含和最大包含

显然,  $\bar{C}X_3=Y_3$ ,  $\bar{C}X_3=Y_2 \cup Y_3 \cup Y_4$ ,  $Bn_C(X_3)=Y_2 \cup Y_4$ , 其中的对象仅根据属性集  $C$  中的属性值是不能确定它们是否属于  $X_3$  的。

#### 4. 信息系统的可描述性

在信息系统  $S$  中, 对于任一  $O$  的子集  $X$ , 如果满足  $\underline{A}X=\bar{A}X$ , 即

$$\bigcup \{s_i \mid s_i \in \Omega \text{ 且 } s_i \subseteq X\} = \bigcup \{s_i \mid s_i \in \Omega \text{ 且 } s_i \cap X \neq \Phi\}$$

则称子集  $X$  是可描述的, 其中  $\Omega=(O, \tilde{S})$  是由划分  $\tilde{S}$  导出的近似空间。  $S$  中所有由划分  $\tilde{S}$  导出的可描述集合的集合记为  $D_S$ 。

一般情况下, 不能根据某些属性随意将对象划分为若干个子集, 除非这些子集是可描述的, 否则将不会有任何意义。因此, 在任何由属性为划分依据的分类系统中, 必须考虑子集的可描述性。

#### 5. 信息系统的表现

设信息系统  $S=\langle O, A, V, \rho \rangle$ , 新的信息系统  $S^*=\langle O_S, A, V, \rho^* \rangle$  称为  $S$  的表现。

其中

$O_S$  是  $S$  的所有基本集的集合, 即  $O_S=\Omega=(O, \tilde{S})$ ;  $\rho^*$  是  $O_S \times A \rightarrow V$  的映射, 对于  $X_i \in O_S$ ,  $a \in A$ ,  $\rho^*(X_i, a)=V$ 。当且仅当所有的  $o \in X_i$  时, 有  $\rho(o, a)=V$ 。

根据上述定义, 合并  $S$  表中所有的重复行, 把对象换成相应的基本集, 就可得到系统  $S$  的表现, 而且  $S^*$  总是可选的信息系统。

**【例 2.6】** 设信息系统  $S$  由表 2.4 给出。

表 2.4 信息系统  $S$ 

$O$	$a_1$	$a_2$	$a_3$	$O$	$a_1$	$a_2$	$a_3$
$o_1$	$B_1$	$C_1$	$D_2$	$o_4$	$B_1$	$C_1$	$D_2$
$o_2$	$B_2$	$C_3$	$D_1$	$o_5$	$B_1$	$C_1$	$D_2$
$o_3$	$B_2$	$C_2$	$D_3$	$o_6$	$B_2$	$C_2$	$D_3$

合并表中重复的行，把对象换成相应的基本集，则得到了  $S$  的表现  $S^*$ ，如表 2.5 所示。

表 2.5  $S$  的表现  $S^*$

$O_S$	$a_1$	$a_2$	$a_3$
$\{o_1, o_4, o_5\}$	$B_1$	$C_1$	$D_2$
$\{o_2\}$	$B_2$	$C_3$	$D_1$
$\{o_3, o_6\}$	$B_2$	$C_2$	$D_3$

此表明确显示，信息系统  $S$  的表现  $S^*$  是一个可选的信息系统。

将信息系统  $S$  转换为它的表现  $S^*$ ，从某种意义上说是一种简化。如果在信息系统的实际开发中采用表现的形式，则不仅可以节省存储空间，而且对数据进行的某些操作的效率也会相应地有所提高。

2.3.5 最简信息系统

1. 属性的依赖关系

设信息系统  $S = \langle O, A, V, \rho \rangle$  中有两个属性  $a_1, a_2$ ，则：

- (1) 若  $\tilde{a}_1 \subset \tilde{a}_2$ ，则在  $S$  中称  $a_2$  依赖于  $a_1$ ，记作  $a_1 \rightarrow a_2$ ；
- (2) 若  $\tilde{a}_1 \not\subset \tilde{a}_2$  和  $\tilde{a}_2 \not\subset \tilde{a}_1$ ，则在  $S$  中称  $a_1, a_2$  相互独立；
- (3) 若  $\tilde{a}_1 = \tilde{a}_2$ ，则在  $S$  中称为  $\tilde{a}_1$  与  $\tilde{a}_2$  等价，记为  $\tilde{a}_1 \sim \tilde{a}_2$ 。

类似地，若  $B, C$  为属性集合  $A$  的子集，则：

- (1) 若  $\tilde{B} \subset \tilde{a}$ ，则说明属性  $a$  依赖属性组  $B$ ，记作  $B \rightarrow a$ ；
- (2) 若  $\tilde{a} \subset \tilde{B}$ ，则说明属性组  $B$  依赖于属性  $a$ ，记作  $a \rightarrow B$ ；
- (3) 若  $\tilde{B} \subset \tilde{C}$ ，则  $B \rightarrow C$ ；
- (4) 若  $\tilde{B} = \tilde{C}$ ，则  $B \sim C$ 。

对于属性集合  $A$  中的任何子集  $X$ ，均有  $A \rightarrow X$ 。

若在  $S$  中有  $B \rightarrow C$ ，则在  $S^*$  中也有  $B \rightarrow C$ 。由于  $S^*$  表要比  $S$  表简单，因此检验  $S^*$  中的依赖关系比检验  $S$  中的依赖关系简单。

根据以上的描述，我们知道信息系统中的某些属性可能是多余的，即它们的值可以由系统中的其他属性的值来确定。

2. 属性组间的相互关系

设在信息系统  $S = \langle O, A, V, \rho \rangle$  中， $B, C$  是从属于  $A$  的属性组。

- (1) 若任一个  $B' \subset B$ ，有  $\tilde{B}' \neq \tilde{B}$ ，则称  $B$  是在  $S$  中独立的属性组。

(2) 若存在一个  $B' \subset B$ , 有  $\tilde{B}' = \tilde{B}$ , 则称  $B$  是在  $S$  中不独立的属性组。

(3) 若  $C \subset B$  且  $\tilde{B} = \tilde{C}$ , 则称  $B$  在  $S$  中可由  $C$  推出。

**【例 2.7】** 设  $S = \langle O, A, V, \rho \rangle$ , 其中  $O = \{o_1, o_2, o_3, o_4, o_5\}$ ,  $A = \{a, b, c, d\}$ 。由这些属性生成的  $O$  的划分为

$$\tilde{a} : \{o_1, o_2, o_5\}, \{o_3, o_4\}$$

$$\tilde{b} : \{o_1\}, \{o_2, o_3, o_4, o_5\}$$

$$\tilde{c} : \{o_1, o_2, o_3, o_4\}, \{o_5\}$$

$$\tilde{d} : \{o_1\}, \{o_3, o_4\}, \{o_2, o_5\}$$

整个属性集  $A$  定义的划分是

$$\tilde{a} \cap \tilde{b} \cap \tilde{c} \cap \tilde{d} = \{o_1\}, \{o_2\}, \{o_3, o_4\}, \{o_5\}$$

分析各个划分, 可知

$$\tilde{d} \subset \tilde{b}, d \rightarrow b$$

$$\tilde{d} \subset \tilde{a}, d \rightarrow a$$

$$\{\tilde{a} \cap \tilde{b} \cap \tilde{c}\} \subset \tilde{d}, \{a, b, c\} \rightarrow d$$

$$\{\tilde{c} \cap \tilde{d}\} \subset (\tilde{a}, \{c, d\}) \rightarrow a$$

$$\{\tilde{c} \cap \tilde{d}\} \subset (\tilde{b}, \{c, d\}) \rightarrow b$$

故

$$\{c, d\} \rightarrow \{a, b\}$$

由上述可知, 设  $B = \{a, b, c\}$ , 则  $B \subset A$ , 且使  $\tilde{B} = \tilde{A}$ , 因此在  $S$  中属性集  $A$  是不独立的。

同样, 由于  $C = \{a, c, d\}$ ,  $D = \{c, d\}$ ,  $C, D \subset A$ , 有  $\tilde{C} = \tilde{D} = \tilde{A}$ 。所以其中  $B$ 、 $D$  在  $S$  中是独立的,  $C$  是不独立的, 因为  $\tilde{C} = \tilde{D}$ 。

属性组的不独立性, 说明在系统中这些属性的值可以从其他属性的值中得到, 即存在依赖关系, 那么怎样衡量属性集之间的依赖关系呢?

### 3. 属性组之间的依赖程度

设信息系统  $S = \langle O, A, V, \rho \rangle$ ,  $B$ 、 $C$  是属性组, 且有  $B, C \subset A$ 。

设

$$\Omega_1 = (O, \tilde{B}); \Omega_2 = (O, \tilde{C}); \text{POS}_B(C) = \bigcup_{X \in \Omega_2} \underline{B}X;$$

则  $\text{POS}_B(C)$  中包含的元素, 是对象集  $O$  中所有根据属性组  $B$  的信息可以准确无误地划分到关系  $\tilde{C}$  的等价类中的对象。

定义属性组  $B$  和  $C$  的依赖程度  $K(B, C)$  为

$$K(B, C) = |\text{POS}_B(C)| / |O|$$

并称属性组  $B$  和  $C$  在信息系统  $S$  中以  $K$  度依赖 ( $0 \leq K \leq 1$ )。

由此可见,  $K(B, C)$  的值将属性组  $B$  和  $C$  的依赖程度量化了, 它计算出根据  $B$  能够被正确分类到  $\tilde{C}$  的等价类中的所有对象在系统中所占的比例, 即属性集  $B$  区分关系  $\tilde{C}$  的等价类的能力。

由定义可知,  $K(B, C) \in [0, 1]$ 。若  $K(B, C) = 0$ , 则表示根据属性集  $B$  的信息无法将任何对象准确地划分到  $\tilde{C}$  的等价类中; 若  $K(B, C) = 1$ , 则表示根据属性集  $B$  的信息可以将任何对象准确地划分到  $\tilde{C}$  的等价类中, 即  $B \rightarrow C$ 。

**【例 2.8】** 假设同例 1.7: 设  $\Omega_1 = (O, \tilde{b}), \Omega_2 = (O, \tilde{c})$ , 则  $\Omega_1 = \{\{o_1\}, \{o_2, o_3, o_4, o_5\}\}, \Omega_2 = \{\{o_1, o_2, o_3, o_4\}, \{o_5\}\}$ 。

取  $X_1 = \{o_1, o_2, o_3, o_4\}$ , 则  $\underline{b}X_1 = \{o_1\}$ ; 取  $X_2 = \{o_5\}$ , 则  $\underline{b}X_2 = \Phi$ ; 故

$$\text{POS}_b(c) = \bigcup_{X \in \Omega_2} \underline{b}X = \{o_1\}$$

$$K(b, c) = |\text{POS}_b(c)| / |O| = 1/5$$

另设  $\Omega_3 = (O, \tilde{d})$ , 则  $\Omega_3 = \{\{o_1\}, \{o_3, o_4\}, \{o_2, o_5\}\}$ 。

取  $X_1 = \{o_1, o_2, o_3, o_4\}$ , 则  $\underline{d}X_1 = \{o_1, o_3, o_4\}$ ; 取  $X_2 = \{o_5\}$ , 则  $\underline{d}X_2 = \Phi$ ; 故

$$\text{POS}_d(c) = \bigcup_{X \in \Omega_2} \underline{d}X = \{o_1, o_3, o_4\}$$

$$K(d, c) = |\text{POS}_d(c)| / |O| = 3/5$$

#### 4. 属性的重要度

设信息系统  $S = \langle O, A, V, \rho \rangle$ ,  $B, C$  是属性组, 且有  $B, C \subset A$ 。取任一  $a \subset B$ , 可定义属性  $a$  关于属性组  $B$  和  $C$  的重要度:

$$\text{SGF}(a, B, C) = K(B, C) - K(B - \{a\}, C)$$

由定义可知,  $\text{SGF}(a, B, C)$  表示的是在由  $\tilde{C}$  生成的划分中,  $B$  中属性  $a$  的重要度。 $\text{SGF}(a, B, C) \in [0, 1]$ 。若  $\text{SGF}(a, B, C) = 0$ , 则表示属性  $a$  关于  $C$  是可省的, 即从属性组  $B$  中除去属性  $a$  后, 原来根据  $B$  可被正确划分到  $\tilde{C}$  的等价类中的所有对象, 根据  $B - \{a\}$  中的信息仍能被准确划分到  $\tilde{C}$  的各等价类中去; 若  $\text{SGF}(a, B, C) \neq 0$ , 则表示属性  $a$  在  $B$  中对于  $C$  是不可缺少的, 即去掉属性  $a$  后, 某些对象将不能被准确地划分了。

依次类推,  $\text{SGF}(a, A, A) = 1 - K(A - \{a\}, A)$  表示的是属性  $a$  在信息系统  $S = \langle O, A, V, \rho \rangle$  中的重要度。

#### 5. 最小独立属性组

设  $S = \langle O, A, V, \rho \rangle$  是一个信息系统, 属性组  $A' \subset A$ , 若  $\tilde{A}' = \tilde{A}$ , 且不存在  $A'$  的子集  $B$  使得  $\tilde{B} = \tilde{A}'$ , 则称  $A'$  为  $A$  的化简, 或  $A'$  是  $S$  中的最小独立属性组。

若  $A'$  是  $S$  中的最小独立属性组, 则对任意  $a \in A$ , 有  $A' \rightarrow a$ 。

作为  $S$  中的最小独立属性组  $A'$ , 它具有与  $A$  同样的划分等价类的能力。换句话说, 对任意对象而言, 若根据  $A$  的信息可被划分到某个等价类中的话, 则根据  $A'$  中较少的信息它仍能被准确地划分到该等价类中。此外,  $A'$  是“最小的”是指, 对于任意的  $B \subset A'$ , 都不具有与  $A'$  同样的划分等价类的能力。值得注意的是,  $S$  中的最小独立属性组不是唯一的。

## 6. 最简信息系统

设  $S = \langle O, A, V, \rho \rangle$  是一个信息系统, 属性组  $A' \subset A$  且是  $S$  中的最小独立属性组, 则相应的信息系统  $S' = \langle O, A', V, \rho' \rangle$  称为最简信息系统, 其中  $\rho'$  是  $\rho$  在集合  $O \times A'$  上的那个部分映射。

由这个定义可以推出, 一个信息系统可以导出多个最简信息系统。例如, 例 1.7 中的  $B$  和  $D$  都是最简信息系统的属性集合。

最简信息系统具有如下几点性质:

- (1) 若  $S$  是最大系统, 则它也是最简系统 (逆命题不成立)。
  - (2) 若  $S$  是最简系统, 则它的所有属性均两两独立 (逆命题不成立)。
  - (3) 具有同样对象集合  $O$  的两个系统  $S$  和  $S_1$ , 若  $\tilde{S} = \tilde{S}_1$ , 则  $S \sim S_1$ 。
- 对于任意信息系统  $S$  必定存在一个与之等价的  $S_1$ 。
- (4) 若信息系统  $S$  是最简的, 则它的表现  $S^*$  也是最简的。

**【例 2.9】** 设  $S = \langle O, A, V, \rho \rangle$ , 且有:

$O$	$a_1$	$a_2$
$o_1$	$e_1$	$f_1$
$o_2$	$e_2$	$f_1$
$o_3$	$e_1$	$f_2$

该系统是最简信息系统, 但由于当  $\rho'(a_1) = e_2, \rho'(a_2) = f_2$  时,  $O_{\rho'} = \Phi$ , 所以  $S$  不是最大系统。

**【例 2.10】** 设  $S = \langle O, A, V, \rho \rangle$ , 且  $O = \{o_1, o_2, o_3, o_4\}, A = \{a_1, a_2, a_3\}$ , 这些属性所确定的划分为

$$\begin{aligned}\tilde{a}_1 &: \{o_1, o_2\}, \{o_3, o_4\} \\ \tilde{a}_2 &: \{o_1\}, \{o_2, o_3, o_4\} \\ \tilde{a}_3 &: \{o_2\}, \{o_1, o_3, o_4\}\end{aligned}$$

根据独立性的定义可知,  $a_1, a_2, a_3$  是两两独立的。但根据化简的定义可知,  $\{a_1,$

$a_2\}$ 、 $\{a_1, a_3\}$ 、 $\{a_2, a_3\}$  都是  $A$  的化简。

在信息系统理论中,属性化简的思想在实践中是非常重要的,这意味着可以从较少的属性中获得同样多的信息。那么,如何对一个特定的信息系统进行化简,求其最简的信息系统呢?下面介绍两种算法。

### (1) 穷举法

穷举法的思想是对属性的所有组合进行全面的搜索,以寻求所有的最小独立属性组,最终达到化简的目的。

设信息系统  $S = \langle O, A, V, \rho \rangle$ , 其中  $A = \{a_1, a_2, \dots, a_n\}$ 。

① 第一步:计算属性组  $A$  的划分  $\tilde{A}$ 。

设  $i=n-1, m=1$ 。

② 第二步:从  $n$  个属性中选出  $C_n^i$  个不同的属性组,记为  $B_1, B_2, \dots, B_i$ , 每个属性组包含有  $i$  个属性,分别计算  $B_j$  ( $j=1, \dots, i$ ) 的划分  $\tilde{B}_j$ 。

如果  $\tilde{B}_j = \tilde{A}$ , 则属性组  $C_m = B_j, m=m+1$ ;

令  $i=i-1$ , 如果  $i>0$ , 则重复第二步, 否则转入第三步。

③ 第三步:在  $C_j$  ( $j=1, \dots, m$ ) 中选出最小独立属性组输出, 算法结束。

从以上的算法步骤可以看出,该算法是一个 NP 问题,计算过程相当复杂烦琐。当然,该算法可以得到进一步的改进,但对于大型的信息系统来说,这种算法还是不太适用的。

在大多数应用中,并没有必要找到所有的最小独立属性组。用户可根据不同的原则来选择一个他认为最好的最小独立属性组,比如选择具有最少属性个数的最小独立属性组;或者根据每个属性的实际含义,为其定义一个费用函数,从而选择具有最小费用的最小独立属性组等。

下面介绍另一种求解最小独立属性组的算法,它只能求得一个最小独立属性组,但这个最小独立属性组是具有最少属性个数的最小独立属性组。

### (2) 基于用户的多项式法

该算法称为基于用户的多项式法,是因为它可以在多项式的时间内得到一个基于用户的最小独立属性组。说它是基于用户的,是指用户可以强制它包含用户所感兴趣的属性,而不管属性是否冗余,这样做也是为了符合实际情况的需要。

该算法利用了属性的重要度和属性之间的依赖程度的概念。下面我们介绍该算法的具体内容。

设信息系统  $S = \langle O, A, V, \rho \rangle$ , 其中  $A = \{a_1, a_2, \dots, a_n\}$ , 用户指定感兴趣的属性集  $\text{INTEREST} = \Phi$ 。



到最小独立属性组后，就可以从原信息系统中删除多余的属性，从而达到简化系统的目的。此外，该算法中的集合变量  $M$  的最终结果实际上是所有最小独立属性交集，它在机器学习中有很大用处，这里就不再进行解释，有兴趣的读者可[有关资料](#)。

**【例 2.11】** 条件同例 2.7。

设  $S = \langle O, A, V, \rho \rangle$ , 其中  $O = \{o_1, o_2, o_3, o_4, o_5\}$ ,  $A = \{a, b, c, d\}$ 。由这些属性生成的  $O$  的划分为

$$\begin{aligned} \tilde{a} &: \{o_1, o_2, o_5\}, \{o_3, o_4\} \\ \tilde{b} &: \{o_1\}, \{o_2, o_3, o_4, o_5\} \\ \tilde{c} &: \{o_1, o_2, o_3, o_4\}, \{o_5\} \\ \tilde{d} &: \{o_1\}, \{o_3, o_4\}, \{o_2, o_5\} \end{aligned}$$

整个属性集  $A$  定义的划分是

$$\tilde{a} \cap \tilde{b} \cap \tilde{c} \cap \tilde{d} = \{o_1\}, \{o_2\}, \{o_3, o_4\}, \{o_5\}$$

用基于用户的多项式法求解最小独立属性组，以化简该系统。

求解过程：

设  $\Phi$ ;

$\Phi$ ;

$\Phi$ ;

$(\quad, \quad, \quad) \quad$  ;

$(\quad, \quad, \quad) \quad$  ;

$(\quad, \quad, \quad) \quad$  . ;

$(\quad, \quad, \quad) \quad$  ;

$\quad$  ;

$\quad$  ;

$\quad$  .  $\neq$  ;

$(\quad, \quad, \quad, \quad) \quad$  . ;

$(\quad, \quad, \quad, \quad) \quad$  . ;

$(\quad, \quad, \quad, \quad) \quad$  . ;

取属性  $\quad$  ;

$\quad$  ;

$\quad$  ;

$\quad$  ;

.....

$\quad$  ;

求得一个最小独立属性组为  $\{c, d\}$ ，即  $\{c, d\}$  是  $A$  的化简。

2.3.6 理想信息系统

上面已经介绍了最大的信息系统、可选的信息系统、最简信息系统的概念，下面定义一个理想信息系统的概念。所谓理想信息系统，是指该系统既是最大的，又是可

选的, 还是最简的, 换句话说, 这种信息系统既无非空信息, 又无多余属性, 且  $O$  中信息为  $\rho'$  的“对象”只有一个。由信息系统的性质可知, 如果一个信息系统既是最大的, 又是可选的, 则它一定是理想的。

我们引入的理想信息系统的概念是一个理想化的概念, 它包含的新系统应该是最完备, 且无冗余的。但正如最大的信息系统一样, 理想信息系统不一定存在, 而且有的系统中保持一定的冗余也有其实际的意义。因此, 仅以理想信息系统作为一个评价系统的标准, 而不要求所有信息系统都能达到理想信息系统的要求。

### 2.3.7 信息系统中的决策规则

在信息系统  $S = \langle O, A, V, \rho \rangle$  中, 将属性集合  $A$  中的属性划分成两个不相交的子集  $C$  和  $D$ , 即  $A = C \cup D$ , 且  $C \cap D = \Phi$ , 则该信息系统可以视为一个决策表  $S = \langle O, C \cup D, V, \rho \rangle$ , 其中  $C$  为条件属性,  $D$  为决策属性。

当且仅当  $C \rightarrow D$  时, 称决策表  $S = \langle O, C \cup D, V, \rho \rangle$  是确定性的, 否则  $S$  是不确定性的。在确定性的决策表中, 当条件属性的值确定时, 决策属性的值也就被唯一地确定下来了。但在不确定性的决策表中, 条件属性的值不能唯一确定所有决策属性的值, 而可能只有一部分的决策属性被条件属性确定, 或完全不能确定。

从决策表中可以推导出一些决策规则。设  $\Omega_1 = (O, \tilde{C})$ ,  $\Omega_2 = (O, \tilde{D})$ , 任取  $X_i \in \Omega_1, Y_j \in \Omega_2$ , 则

(1) 当  $X_i \cap Y_j \neq \Phi$  时, 有

$$r_{ij} : \text{Des}(X_i) \rightarrow \text{Des}(Y_j)$$

其中,  $\text{Des}(X_i)$  和  $\text{Des}(Y_j)$  分别是等价类  $X_i$  和等价类  $Y_j$  中的对象的特性描述。

① 当  $X_i \cap Y_j = X_i$  时 ( $X_i$  完全被  $Y_j$  包含), 建立的规则  $r_{ij}$  是确定的, 规则可信度  $cf = 1.0$ 。

② 当  $X_i \cap Y_j \neq X_i$  时 ( $X_i$  部分被  $Y_j$  包含), 建立的规则  $r_{ij}$  是不确定的, 规则可信度为:

$$cf = \frac{|X_i \cap Y_j|}{|X_i|}$$

(2) 当  $X_i \cap Y_j = \Phi$  时 ( $X_i$  不被  $Y_j$  包含),  $X_i$  和  $Y_j$  不能建立规则。

**【例 2.12】** 假设同例 2.5, 从例 2.5 中可以得出以下规则。

$$\Omega_1 = (O, \tilde{C}) = \{Y_1, Y_2, Y_3, Y_4\}:$$

$$\text{Des}(Y_1) = (a_1 = e_{11}, a_2 = e_{20})$$

$$\text{Des}(Y_2) = (a_1 = e_{10}, a_2 = e_{21})$$

$$\text{Des}(Y_3) = (a_1 = e_{12}, a_2 = e_{20})$$

$$\text{Des}(Y_4) = (a_1 = e_{11}, a_2 = e_{21})$$

$$\Omega_2 = (O, \tilde{D}) = \{X_1, X_2, X_3\}:$$

$$\text{Des}(X_1) = (a_3 = e_{32})$$

$$\alpha_C(X_1) = 1.0$$

$$\text{Des}(X_2) = (a_3 = e_{31})$$

$$\alpha_C(X_2) = 0.5$$

$$\text{Des}(X_3) = (a_3 = e_{30})$$

$$\alpha_C(X_3) = 0.5$$

(1) 把  $Y$  作为前提、 $X_1$  作为结论，可以得到下列规则。

$$r_{11}: \text{Des}(Y_1) \rightarrow \text{Des}(X_1), \text{即 } (a_1 = e_{11}) \wedge (a_2 = e_{20}) \rightarrow (a_3 = e_{32}) (cf = 1.0)。$$

$$r_{21}: \text{不存在, 因为 } Y_2 \cap X_1 = \Phi。$$

$$r_{31}: \text{不存在, 因为 } Y_3 \cap X_1 = \Phi。$$

$$r_{41}: \text{不存在, 因为 } Y_4 \cap X_1 = \Phi。$$

因为  $\alpha_C(X_1) = 1.0$ ，所以根据划分  $\tilde{C}$  判断  $O$  中的对象是否属于  $X_1$  类的可信度为 1.0。

(2) 把  $Y$  作为前提、 $X_2$  作为结论，可以得到下列规则。

$$r_{12}: \text{不存在, 因为 } Y_1 \cap X_2 = \Phi。$$

$$r_{22}: \text{Des}(Y_2) \rightarrow \text{Des}(X_2), \text{即 } (a_1 = e_{10}) \wedge (a_2 = e_{21}) \rightarrow (a_3 = e_{31}) (cf = 0.67)。$$

$$r_{32}: \text{不存在, 因为 } Y_3 \cap X_2 = \Phi。$$

$$r_{42}: \text{Des}(Y_4) \rightarrow \text{Des}(X_2), \text{即 } (a_1 = e_{11}) \wedge (a_2 = e_{21}) \rightarrow (a_3 = e_{31}) (cf = 0.5)。$$

因为  $\alpha_C(X_2) = 0.5$ ，所以根据划分  $\tilde{C}$  判断  $O$  中的对象是否属于  $X_2$  类的可信度为 0.5。

(3) 把  $Y$  作为前提、 $X_3$  作为结论，可以得到下列规则。

$$r_{13}: \text{不存在, 因为 } Y_1 \cap X_3 = \Phi。$$

$$r_{23}: \text{Des}(Y_2) \rightarrow \text{Des}(X_3), \text{即 } (a_1 = e_{10}) \wedge (a_2 = e_{21}) \rightarrow (a_3 = e_{30}) (cf = 0.33)。$$

$$r_{33}: \text{Des}(Y_3) \rightarrow \text{Des}(X_3), \text{即 } (a_1 = e_{12}) \wedge (a_2 = e_{20}) \rightarrow (a_3 = e_{30}) (cf = 1.0)。$$

$$r_{43}: \text{Des}(Y_4) \rightarrow \text{Des}(X_3), \text{即 } (a_1 = e_{11}) \wedge (a_2 = e_{21}) \rightarrow (a_3 = e_{30}) (cf = 0.5)。$$

因为  $\alpha_C(X_3) = 0.5$ ，所以根据划分  $\tilde{C}$  判断  $O$  中的对象是否属于  $X_3$  类的可信度为 0.5。

### 2.3.8 不确定性信息系统

#### 1. 不确定性信息系统的定义

一般来说，一个信息系统对对象的描述是确定的。这也就是说，对于信息系统中的一个给定的对象和给定的属性（属性-值对），不存在该对象是否拥有该属性值的不确定性，即属性值是确定的。然而，这种确定性也是一种限制，它在两个方面限制了

表达能力：其一、世界上的所有对象必须采用统一的表达方式；其二、由于对象的描述是确定的，因而无法给出对象属性的表达程度，一个对象对某一属性值只能有“拥有”或“不拥有”两种状态。

为了能表达对象的不确定性和不同的重要度，基于 2.2 节中给出的信息系统的数学模型引入了不确定性信息系统的概念。在不确定性信息系统中，每一个对象设有一个不确定性值  $u$  和重要度  $d$ 。不确定性值  $u$  是一个  $[0.0, 1.0]$  范围内的实数，当  $u$  为 1.0 时，表示一个完全肯定的对象；当  $u$  为 0.0 时，表示一个完全否定的对象。重要度  $d$  表示一个对象在信息系统中的重要性。下面给出不确定性信息系统（UIS）的定义：

$UIS = \langle O, C \cup D, \{V_a\}_{a \in C}, u, d \rangle$  是不确定性信息系统。

其中， $O$  是非空的对象集合；

$C$  是非空的条件属性的集合；

$D$  是带有不确定值  $u$  的决策属性；

$V_a$  是条件属性“ $a$ ”的定义域，它至少包含两个元素。

每一个从属于  $C$  的条件属性  $a$  均可被视为是一个函数，对每个对象  $o \in O$  赋值  $a(o) \in V_a$ 。每一个从属于  $O$  的对象都有条件属性  $C$  的一组确定的值、决策属性  $D$  的不确定的值和对象重要度  $d$  的实数值。

**【例 2.13】** 表 2.6 给出了一个不确定性信息系统的例子。

表 2.6 不确定性信息系统

$O$	$c_1$	$c_2$	dec	$d$
$o_1$	0	0	0.75	4
$o_2$	0	1	0.67	3
$o_3$	0	2	0.35	4
$o_4$	1	0	0.75	4
$o_5$	1	1	0.67	3
$o_6$	1	2	0.35	4

如表中所示，对象集合  $O = \{o_i\} (i=1, 2, \dots, 6)$ ，条件属性的集合  $C = \{c_1, c_2\}$ ， $V_{c_1} = \{0, 1\}$ ， $V_{c_2} = \{0, 1, 2\}$ ，决策属性  $D = \{\text{dec}\}$  是一组不确定的值， $\{u_{\text{dec}_i}\} = \{0.75, 0.67, 0.35, 0.75, 0.67, 0.35\} (i=1, 2, \dots, 6)$ ，每个对象设有一个重要度  $d$ ，其值为  $\{d(\text{obj}_i)\} = \{4, 3, 4, 4, 3, 4\} (i=1, 2, \dots, 6)$ 。

## 2. 不确定性信息系统中的噪声控制

在现实世界中，信息系统中的每一类对象（肯定对象和否定对象）均可能包含不同的噪声。为了对不确定性信息系统中的噪声进行控制，引入了相对分类错误的概念。

它的主要思想是：根据某种分类原则，将对象划分到肯定区域、边界区域和否定区域三个部分中去，从而产生一些几乎在任何情况下都是正确的规则。为此，引入了两个分类因子  $P_\beta$  和  $N_\beta$  ( $0.0 \leq P_\beta, N_\beta \leq 1.0$ )， $P_\beta$  和  $N_\beta$  可以同时存在也可以取相同的值，它们分别由肯定区域和否定区域中测量的噪声度来确定。

设  $X$  为对象集合  $O$  的非空子集，将集合  $X$  中的对象错误划分到肯定对象类  $P_{\text{class}}$  和否定对象类  $N_{\text{class}}$  的相对概率定义如下：

$$C_P(X) = \frac{\sum (d_i \times (1 - u_i))}{\sum d_i} \quad o_i \in X, X \subseteq O$$

$$C_N(X) = \frac{\sum (d_i \times u_i)}{\sum d_i} \quad o_i \in X, X \subseteq O$$

其中， $\sum d_i$  是集合  $X$  中所有对象的重要度的和； $\sum (d_i \times u_i)$  是集合  $X$  中对象的重要度  $d_i$  和决策属性的不确定性值  $u_i$  的乘积的和； $\sum (d_i \times (1 - u_i))$  是集合  $X$  中对象的重要度  $d_i$  和  $1 - u_i$  的乘积的和。如果将对象划分到肯定对象类中，则错误率为  $C_P(X)$ ；如果将对象划分到否定对象类，则错误率为  $C_N(X)$ 。

基于相对分类错误的概念，当且仅当分类错误率  $C_P(X)$  小于等于给定的精确度  $P_\beta$  时，对象集合  $X$  属于肯定类；或者当且仅当分类错误率  $C_N(X)$  小于等于给定的精确度  $N_\beta$  时，对象集合  $X$  属于否定类，即

$$P_{\text{class}} \supseteq X \quad \text{当且仅当 } C_P(X) \leq P_\beta$$

$$N_{\text{class}} \supseteq X \quad \text{当且仅当 } C_N(X) \leq N_\beta$$

否则，对象集合  $X$  属于边界区域。

**【例 2.14】** 假设同例 2.13，并设  $P_\beta = 0.30$ ， $N_\beta = 0.6$ ，对象集合  $X_1 = \{o_1\}$ ， $X_2 = \{o_2\}$ ， $\dots$ ， $X_6 = \{o_6\}$ ，则

$$C_P(X_1) = \frac{4 \times (1.0 - 0.75)}{4} = 0.25 \quad C_N(X_1) = \frac{4 \times 0.75}{4} = 0.75$$

$$C_P(X_2) = \frac{3 \times (1.0 - 0.67)}{3} = 0.33 \quad C_N(X_2) = \frac{3 \times 0.67}{3} = 0.67$$

$$C_P(X_3) = \frac{4 \times (1.0 - 0.35)}{3} = 0.65 \quad C_N(X_3) = \frac{4 \times 0.35}{3} = 0.35$$

类似的

$$C_P(X_4) = 0.25, C_N(X_4) = 0.75;$$

$$C_P(X_5) = 0.33, C_N(X_5) = 0.67;$$

$$C_P(X_6) = 0.65, C_N(X_6) = 0.35$$

故

$$P_{\text{class}} = X_1 \cup X_4 = \{o_1, o_4\}$$

$$P_{\text{class}} = X_3 \cup X_6 = \{o_3, o_6\}$$

不确定性信息系统的其他理论, 这里不再介绍, 感兴趣的读者可参看相应的文献资料。下面仍针对前面所述的确定的信息系统进行论述。

## 2.4 信息系统的连接

在实际工作中, 仅有一个信息系统的信息往往不能满足用户的需要, 而且若干个信息系统之间往往还存在有某种特定的联系。因此, 将若干个信息系统  $S_i$  ( $i=1, \dots, K$ ) 连接成一个“统一”的信息系统  $S$ , 在现实应用中是非常有意义的。下面, 将对信息系统的连接及其操作的有关内容进行介绍。

### 2.4.1 良好的连接系统

设  $S = \langle O, A, V, \rho \rangle$  和  $S_i = \langle O_i, A_i, V_i, \rho_i \rangle$  ( $i = 1, \dots, K$ ) 都是信息系统, 如果

$$O = \bigcup_{i=1}^K O_i, A = \bigcup_{i=1}^K A_i, V = \bigcup_{i=1}^K V_i$$

$$\rho/O_i \times A_i = \rho_i (i = 1, 2, \dots, K)$$

$$\rho_o = \bigcup_{i=1}^K \rho_{o_i}, o \in O$$

则称  $S$  是  $S_i$  ( $i=1, \dots, K$ ) 的连接, 并记为  $S = \bigcup_{i=1}^K S_i$ 。

对于一个连接系统  $S = \bigcup_{i=1}^K S_i$ , 如果其满足下面两个条件:

(1) 若  $(O_i \cap O_j) \neq \Phi$ , 且  $(A_i \cap A_j) \neq \Phi$ , 则

$$\rho_i/(O_i \cap O_j) \times (A_i \cap A_j) = \rho_j/(O_i \cap O_j) \times (A_i \cap A_j)$$

对所有的  $i=1, \dots, K$  都成立;

(2)  $\rho_o = \bigcup_{i=1}^K \rho_{o_i}$  对所有的  $o \in O$  及  $a \in A$  都有定义。

则称该信息系统为良好的连接系统。

良好的连接系统需要满足的两个条件是非常明显的。第一个条件是保证各个信息系统连接时数据的一致性; 第二个条件就是要求连接起来的系统的所有属性的值都应有定义, 这才能保证函数  $\rho$  是完全的。

一般地说, 必须使一个连接系统是良好的, 否则连接起来的系统是无法工作的。下面举一例进行说明。

【例 2.15】 试将表 2.7、表 2.8 所示的两个信息系统  $S_1$  和  $S_2$  连接起来。

表 2.7  $S_1$  的函数  $\rho_1$

$O$	$a$	$b$	$c$
$o_1$	$e_1$	$f_1$	$r_2$
$o_2$	$e_1$	$f_2$	$r_1$
$o_3$	$e_2$	$f_1$	$r_2$
$o_4$	$e_2$	$f_1$	$r_2$

表 2.8  $S_2$  的函数  $\rho_2$

$O'$	$c$	$d$	$e$
$o_3$	$r_2$	$h_1$	$k_1$
$o_4$	$r_2$	$h_2$	$k_1$
$o'_1$	$r_1$	$h_3$	$k_1$
$o'_2$	$r_2$	$h_1$	$k_2$

连接的结果如表 2.9 所示。

表 2.9  $S=S_1 \cup S_2$  的函数  $\rho$

$OUO'$	$a$	$b$	$c$	$d$	$e$
$o_1$	$e_1$	$f_1$	$r_2$	—	—
$o_2$	$e_1$	$f_2$	$r_1$	—	—
$o_3$	$e_2$	$f_1$	$r_2$	$h_1$	$k_1$
$o_4$	$e_2$	$f_1$	$r_2$	$h_2$	$k_1$
$o'_1$	—	—	$r_1$	$h_3$	$k_1$
$o'_2$	—	—	$r_2$	$h_1$	$k_2$

因为某些属性的某些值在表中无意义，即由表确定的函数  $\rho$  不完全，所以根据信息系统的定义，这就不能算是一个完整的信息系统。

2.4.2 连接系统的分类

下面将介绍两种类型的连接系统。

1. 属性连接系统

设  $S = \langle O, A, V, \rho \rangle$  和  $S_i = \langle O_i, A_i, V_i, \rho_i \rangle$  ( $i=1, \dots, K$ ) 都是信息系统，如果  $S = \bigcup_{i=1}^K S_i$  且  $S_i = S/A_i$ ，则称  $S$  为属性连接系统。

【例 2.16】 设  $S_1$ 、 $S_2$  为两个对象集合相同而各属性集合不同的信息系统，如表 2.10、表 2.11 所示。

表 2.10  $S_1$  的函数  $\rho_1$

$O$	$a$	$b$	$c$
$o_1$	$a_1$	$b_1$	$c_2$
$o_2$	$a_1$	$b_2$	$c_1$
$o_3$	$a_2$	$b_1$	$c_2$
$o_4$	$a_1$	$b_1$	$c_2$

表 2.11  $S_2$  的函数  $\rho_2$

$O'$	$a$	$d$	$e$
$o_1$	$a_1$	$d_1$	$e_2$
$o_2$	$a_1$	$d_2$	$e_1$
$o_3$	$a_2$	$d_1$	$e_1$
$o_4$	$a_1$	$d_1$	$e_2$



则  $S_1$ 、 $S_2$  的连接结果如表 2.12 所示。

表 2.12  $S=S_1 \cup S_2$  的函数  $\rho$  (属性连接)

$OUO'$	$a$	$b$	$c$	$d$	$e$
$o_1$	$a_1$	$b_1$	$c_2$	$d_1$	$e_2$
$o_2$	$a_1$	$b_2$	$c_1$	$d_2$	$e_1$
$o_3$	$a_2$	$b_1$	$c_2$	$d_1$	$e_1$
$o_4$	$a_1$	$b_1$	$c_2$	$d_1$	$e_2$

由该例可知，如果被连接的各个信息系统具有相同的对象集合，而各属性集合不同，则这种连接就是属性连接。

2. 对象连接系统

设  $S = \langle O, A, V, \rho \rangle$  和  $S_i = \langle O_i, A_i, V_i, \rho_i \rangle$  ( $i=1, \dots, K$ ) 都是信息系统，如果  $S = \bigcup_{i=1}^K S_i$  且  $S_i = S/O_i$ ，则称  $S$  为对象连接系统。

【例 2.17】 设  $S_1$ 、 $S_2$  为两个对象集合不同而各属性集合相同的信息系统，如表 2.13、表 2.14 所示。

表 2.13  $S_1$  的函数  $\rho_1$

$O$	$a$	$b$	$c$
$o_1$	$a_1$	$b_1$	$c_2$
$o_2$	$a_2$	$b_2$	$c_2$
$o_3$	$a_1$	$b_2$	$c_1$
$o_4$	$a_1$	$b_1$	$c_1$

表 2.14  $S_2$  的函数  $\rho_2$

$O'$	$a$	$b$	$c$
$o_3$	$a_1$	$b_2$	$c_1$
$o_4$	$a_1$	$b_1$	$c_1$
$o'_1$	$a_2$	$b_2$	$c_2$
$o'_2$	$a_1$	$b_2$	$c_1$

则  $S_1$ 、 $S_2$  的连接结果如表 2.15 所示：

表 2.15  $S=S_1 \cup S_2$  的函数  $\rho$  (对象连接)

$OUO'$	$a$	$b$	$c$	$OUO'$	$a$	$b$	$c$
$o_1$	$a_1$	$b_1$	$c_2$	$o_4$	$a_1$	$b_1$	$c_1$
$o_2$	$a_2$	$b_2$	$c_2$	$o'_1$	$a_2$	$b_2$	$c_2$
$o_3$	$a_1$	$b_2$	$c_1$	$o'_2$	$a_1$	$b_2$	$c_1$

从该例可知，如果被连接的各个信息系统具有相同的属性集合，而各个对象集合不同，则这种连接就是对象连接。

从上面的描述中不难理解：无论属性连接，还是对象连接都是非常有意义的。例如，某公司管理中有两个信息系统，一个是人事管理信息系统，另一个是工资管理

信息系统, 它们拥有相同的对象集合 (公司职员), 但属性集合却不同, 当主管人员希望同时了解公司职员中人事与工资的整体情况时, 可以把两个系统连接成一个属性连接的总的信息系统, 从而简化操作。又如, 同一个保险公司, 在各区都设有信息系统, 这些信息系统的属性集合都是相同的, 而对象集合则不同 (各区的客户), 也可以把它们连接成一个对象连接的总的信息系统。

### 2.4.3 连接系统的基本性质

设  $S = \langle O, A, V, \rho \rangle$  和  $S_i = \langle O_i, A_i, V_i, \rho_i \rangle (i = 1, \dots, K)$  都是信息系统, 且  $S = \bigcup_{i=1}^K S_i$ ,

则连接系统  $S$  有如下几个基本性质。

(1) 如果  $S$  为属性连接系统, 即使每个  $S_i$  都是最简系统, 则  $S$  也不一定是最简系统。

(2) 如果  $S$  为对象连接系统, 且每个  $S_i$  都是最简系统, 则  $S$  也是最简系统。

(3) 如果  $S$  为属性连接系统, 且每个  $S_i$  都是可选的, 则  $S$  也是可选的。

(4) 如果  $S$  为对象连接系统, 即使每个  $S_i$  都是可选的, 则  $S$  也不一定是可选的。

(5) 如果  $S$  为属性连接系统, 即使每个  $S_i$  都是最大的, 则  $S$  也不一定是最大的。

(6) 如果  $S$  为对象连接系统, 且每个  $S_i$  都是最大的, 则  $S$  也是最大的。

(7) 如果  $S$  为对象连接系统, 则对于所有的  $\rho' \in \text{INF}(S)$ , 有  $o_{\rho'} = \bigcup_{i=1}^K o_{i_{\rho'}}$ 。

信息系统的连接是系统的进一步扩展, 而连接系统的基本性质是扩展的依据。

## 习 题

1. 什么样的关系可以称为等价关系?
2. 什么是最大的信息系统?
3. 最简信息系统具有什么性质?
4. 什么是良好连接的系统?

# 第 3 章 信息系统的开发

建设一个工程需要科学的方法支持，同样构建信息系统也需要与之特性相适应的科学方法。目前，普遍认为信息系统的开发过程由系统规划、系统分析、系统设计、系统实现等步骤组成，有结构化、面向对象及原型化等许多开发方法。这些方法有各自的特点和适用范围。在开发信息系统之初，必须首先确定采用什么样的开发方法来指导信息系统的开发，这对信息系统的开发工作有着重要的意义。

## 3.1 信息系统生命周期

信息系统从提出需求、形成概念开始，经过分析论证、系统开发、使用维护，直到淘汰或被新的信息系统所取代的全过程称为信息系统生命周期（Information Systems Life Cycle）。就像人的一生，要经历婴幼儿期、青少年、成年、壮年、老年直至死亡一样，一个信息系统同样存在着从产生、发展到死亡的过程。信息系统生命周期可以按照系统开发活动的需要，划分为若干个阶段。目前，各阶段的划分尚未统一，但一般认为，信息系统生命周期可分为如下五个阶段，如图 3.1 所示。

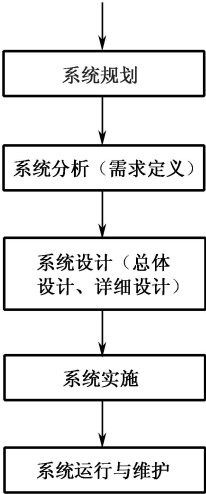


图 3.1 信息系统生命周期

- (1) 系统规划
- (2) 系统分析
- (3) 系统设计
- (4) 系统实施
- (5) 系统运行与维护

其中，系统开发包括前四个阶段的工作。下面简述各阶段的主要内容。

## 1. 系统规划

系统规划的主要任务就是提出信息系统建设的设想，进行可行性研究，并对将要开发的信息系统进行战略规划。

可行性研究是在系统开发项目确定之前，对系统开发的必要性和可能性及可能的候选方案，从整个系统生命周期的角度进行分析和评价，为上级主管部门进行决策提供科学依据。

当一个部门的现行信息系统因种种原因已经不能满足要求，提出建立新的计算机信息系统或改进现有的系统时，系统开发就开始了。可行性研究包括明确任务、环境调查、提出方案和可行性分析。

(1) 明确任务。明确任务主要从系统开发的角度，对用户设想的目标、新系统的功能范围及基本工作过程，以及其他关键性问题做出明确的描述。

(2) 环境调查。环境调查的目的是对系统所在的环境给出一个概括性说明，以便提出候选方案和进行可行性分析。重点调查组织结构、现行系统等的情况，包括现行系统存在的主要问题及其他重要因素。

(3) 提出方案。在环境调查的基础上，根据用户提出的要求，对开发新系统的需求做出分析和预测，同时考虑建设新系统所受到的各种制约因素，根据需求和可能，给出几种拟建系统的候选规模及方案。

(4) 可行性分析。可行性分析是对拟建系统的各种候选方案在技术上、经济上、运行上是否可行进行分析，并在可行性分析的基础上，对各种候选方案进行比较，给出建设性的结论。

信息系统的战略规划将在第4章详细介绍。

## 2. 系统分析

系统分析的目的是解决“做什么”的问题，它是在可行性分析的基础上，针对现行系统进行全面的调查分析，从而提出新系统的逻辑模型。系统分析包括需求调查、数据分析、功能分析和系统定义等方面。

(1) 需求调查。可行性研究阶段的环境调查的重点是了解这个项目是否有必要,而需求调查则是为了弄清现行系统的基本功能及信息流程,以便在此基础上提出新系统的逻辑模型,其重点在于信息系统的内部结构、具体功能、组织安排、先后次序等,这些正是在新系统中有可能要加以修改和变更的内容。因此,其工作的细致程度较环境调查要高得多,工作量也大得多,人力投入也要大得多。

(2) 数据分析。数据分析是对调查得到的大量材料,进行整理、分类、汇总、分析和归纳。它采用规范的工具和方法,弄清信息系统中各类数据的属性、数据的存储要求、数据的查询要求等,并给出定性和定量的描述和分析。

(3) 功能分析。功能分析采用规范的工具和方法,精确地描述用户要求处理过程“做什么”。其中最基本的部分是处理的逻辑,即用户对这个处理过程的逻辑要求及该过程的输出数据流与输入数据流之间所具有的逻辑关系。

(4) 系统定义。系统定义是指在逻辑上定义新系统,即提出新系统的逻辑模型。

### 3. 系统设计

系统设计就是为实现系统分析所提出的逻辑模型所作的各种技术考虑与设计,它根据新系统的逻辑模型建立系统的物理模型,也即根据新系统逻辑功能的要求,考虑系统的规模和复杂程度等实际条件,进行若干具体设计,确定系统的实施方案,从而解决系统“怎么做”的问题。系统设计包括模块设计、代码设计、输入/输出设计、数据库设计和可靠性设计等。

(1) 模块设计。模块设计又称系统控制结构设计,它主要对系统内部进行层次分解,划分系统的模块结构,并确定模块的调用和模块之间数据流与控制流的传递关系。

(2) 代码设计。代码设计包括确定代码的对象、名称、目的、使用范围、数量、编码方法、编码构成等内容,并编写代码对照表。

(3) 输入/输出设计。输入/输出设计即系统人机界面设计,它包括输入/输出方法的选择、输入/输出设备的选择、输入/输出格式设计及输入有效性检查等。

(4) 数据库设计。大中型信息系统由于对时间和空间的要求比较高,处理过程也比较复杂,因此必须建立在数据库系统之上。数据库设计是在需求分析的基础上,确定数据的存储方法和存储结构,进行满足应用要求及符合语义的逻辑设计,进行具有合理的存储结构的物理设计,实现数据库的运行等。

(5) 可靠性设计。可靠性设计包括系统的安全保密性能设计、系统与文件的备份与恢复等。

## 4. 系统实现

系统实现是真正解决“具体做”的问题，它是新系统付诸实施的实施阶段。系统实现阶段具体实现系统设计阶段的新系统的物理模型。它主要包括软、硬件准备，程序设计，数据收集与准备，人员培训，系统测试，系统转换（新旧系统转轨），系统评价等。

## 5. 系统运行与维护

系统的验收通过与交付使用，标志着整个系统开发工作的结束。接下来是信息系统生命周期中的最后一个阶段，即系统运行与维护阶段。新系统要具有长久的生命力，就必须不断地完善，以适应变化，这就是系统维护。

当系统不能很好地满足要求时，就可能又提出修改或研制更新系统的要求，于是系统开发的整个过程又要重新开始。

# 3.2 常用信息系统开发方法

常用的信息系统开发方法有结构化开发方法、面向对象开发方法和原型化开发方法。

## 3.2.1 结构化开发方法

随着信息系统开发建设的逐步展开和深入，早期的信息系统开发方法暴露出一些突出的问题，比如工作阶段的划分原则不明确，各阶段的工作缺乏规范的规程、方法、表达工具与标准；系统建设过程中用户参与程度低，用户与专业人员的对话缺乏有效的手段；系统开发的工作任务集中在系统实施阶段，系统分析、设计工作不够深入；系统实施阶段的工作采取“自底向上”的方法，系统总体功能与目标的实现难以保证等。

“结构化”一词在系统建设中的含意是，用一组规范的步骤、准则和工具来进行某项工作。基于信息系统生命周期概念的结构化方法则为信息系统的建设提供了规范的步骤、准则与工具，以弥补传统方法的不足。一般来说，在各类系统的建设中，结构化方法的基本思路大体都是把整个系统的开发过程分成若干阶段，每个阶段进行若干项活动，每项活动应用一系列标准、规范、方法和技术，完成一个或多个任务，形成符合给定规范的产品成果。具体到信息系统的开发中，结构化方法就是按照信息系统

的生命周期阶段划分,采用系统工程的思想 and 工程化方法,按用户至上的原则,模块化、自顶向下对信息系统进行分析和设计。所以,结构化方法有时也叫做生命周期法。

结构化开发方法具有如下一些主要特点。

- (1) 自顶向下整体分析和设计、自底向上逐步实施的系统开发过程。
- (2) 用户至上。开发过程要充分面向用户,了解用户需求。
- (3) 深入调查研究。
- (4) 严格区分工作阶段,各个开发阶段任务明确,文档齐全,开发过程有序。
- (5) 分析和设计必须充分预料可能发生的情况。

但是,需要说明的是,从多年的信息系统开发实践来看,结构化方法能够得以有效运用需要具备几个前提,主要包括:所有的需求能被预先定义;项目参与者之间能够清晰而准确地交流;建立的图形化、表格化模型对应用系统的反映是充分的。

结构化开发方法将在第5章进行详细介绍。

### 3.2.2 面向对象开发方法

面向对象方法学认为,客观世界是由各种各样的对象组成的,每种对象都有各自的内部状态和运动规律,不同对象之间的相互作用和联系就构成了各种不同的系统。面向对象的信息系统开发方法就是基于构造问题域的对象模型,以对象为中心构造信息系统的方法,其基本思想是用对象模拟问题域的实体,以对象之间的联系来刻画实体之间的联系。面向对象开发方法是从20世纪80年代各种面向对象的程序设计语言中逐步发展而来的。

面向对象开发方法是目前比较主流和常用的一种信息系统开发方法,将在第6章进行详细介绍。

### 3.2.3 原型化开发方法

原型化开发方法简称原型法。

#### 1. 原型法的思想

在生命周期法中,要求系统开发人员和用户在系统开发初期就要对整个系统的功能有全面、深刻的认识,并制定出每一阶段的计划和说明书,以后的工作便围绕这些文档进行,即在系统开发初期(系统建成之前),就要预先知道用户的最终要求,然后围绕着这一需求进行下一步的分析与设计。如果用户需求不能被预知或被错误地理解了,那么以后的工作就失去了意义。但是,随着企业自身的调整、新的管理方法的提

出及信息技术的飞速发展，出现了许多新的要求和新的情况，给这种传统的开发方法带来了严峻的挑战。

为了适应竞争，企业的结构和其经营项目在不断变化，这对信息系统提出了更高的要求。

① 信息系统的开发要快。以往的开发方法涉及面太广，人员太多，手续太繁杂，如果还是这样来开发信息系统，那么系统的建成之日可能就是它的淘汰之时。

② 信息系统要有灵活性。由于经营业务的灵活性，信息系统的使用环境经常地发生变化，所以要有足够的灵活性才能保证信息系统的正常运转。传统的设计方法，从一开始就给系统定下了一个框框，系统的一切活动都围绕着这个框框进行，如果出现了不能预料的变化，再来修改就很困难了。

为解决这些问题，考虑到人自身的特点——灵活、多变和依经验办事，产生了一种新的信息系统的开发方法——原型法（Prototyping Method）。这种方法的思想基础是：用户和开发人员之间总是存在着这样或那样的隔阂，用户或者自己也不清楚系统的最终需求，或者由于交流上的障碍而无法把自己的意图向开发人员完全表达出来。用户只有看到一个具体的系统，才能清楚地了解自己的需要和系统的缺点。这说明，并非所有的需求都能预先定义。由于存在的隔阂，系统不能满足用户的要求是常有的事。因此信息系统的开发过程中大量的反复是必要的、不可避免的，也是使系统具有更强的适应性所要求的。

基于上述观点，原型法就产生了与传统开发方法的两个截然不同的特点：

- (1) 在未完全弄清楚需求之前，通过一个原型化设计环境，迅速地建立原始系统；
- (2) 在原型化环境上能方便地对原始系统进行大量的修改、扩充和完善。

## 2. 原型法的开发过程

利用原型法开发信息系统大致要经过下面几个阶段，如图 3.2 所示。

(1) 可行性研究阶段。在讨论是否开发一个系统时，首先要从宏观上对系统开发的意义、费用和时间作粗略的计算，以确定下面的工作是否进行下去。

(2) 确定系统的基本要求阶段。系统开发人员向用户了解用户对信息系统的基本需求，即应该具有的一些基本功能、人机界面的基本形式等。由于原型法的特点，所以不要求开发人员费极大的力气去争取对系统的完全了解，了解可以是不完全的，也可能会有缺陷，但这些在后面几个阶段的工作中是可以发现和改正的。当然，开发人员不能借此敷衍了事，事实上，错误发现得越早，改正的代价就越小。

(3) 建造一个原始系统阶段。在对系统要求有了基本了解的基础上，系统开发人员应争取尽快地建造一个具有这些基本功能的原始系统。在建造原始系统时，要考虑



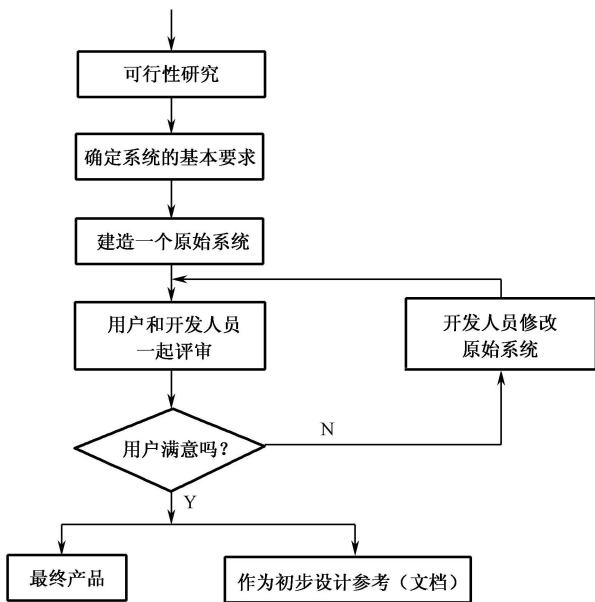


图 3.2 原型法的开发过程

到以后的修改的容易性。由于要求速度快，所以这一阶段应该尽量使用一些软件工具，特别是专门的原型建造工具，辅助进行原始系统的实现。原型法的开发过程非常重视软件工具的使用，只有有效地利用软件工具才能很快地建成一个系统，并能多次对其进行修改、完善。

(4) 用户和开发人员一起评审阶段。这一阶段是整个开发过程的关键。用户和开发人员一起对刚完成的或经过若干次修改后的系统进行评审，提出完善意见。在这个阶段，用户是主角。用户通过亲自使用这个系统，更能了解到自己的需求到底是什么，更能发现系统是否存在问题。这时，开发人员一方面要记录下用户提出的该系统的缺点和不足之处，同时也要借这具体的系统引导，启发用户表达对系统的最终要求，从而清楚地了解用户的意图。

根据心理学家的经验，每个人在借助一个具体事物来表达自己的看法时，比不借助任何具体事物而论述要全面、深刻，也容易得多。用户在一个具体的系统摆在面前时，就能借助这个系统发表自己的观点、意见及以前无法用其他形式向开发人员表达的要求。一个具体的系统，不管存不存在缺陷或不足，常常比一大堆的文件、手册更能说明问题，它是用户和开发人员之间最理想的通信媒介。原型法之所以能得以迅速发展，就是因为它能提供雏形系统供用户和开发人员研究，从而使最终的系统更符合用户的要求。

(5) 开发人员修改原始系统阶段。在评审阶段，用户就实际的系统提供了新的要求，或指出了原始系统中存在的问题，开发人员就要根据用户的意见对原始系统进行修改、扩充和完善。

(6) 结束阶段。开发人员在原始系统进行修改后，又回到第四阶段与用户一起就完成的系统进行评审，如果不满足要求，则要回到第五阶段进行下一轮循环，反复地进行修改和评审。

如果经用户评审，该系统符合要求，则可根据开发原始系统的目的，或者作为最终的信息系统投入正常运行，或者是把该系统作为初步设计的基础，参照这个原始系统设计出实际系统应具有的功能，进行具体设计与实施。

### 3. 原型法的种类与特点

根据原型法的应用目的及场合，可以把它分为以下三种。

#### (1) 丢弃式 (Throw-it-away Prototyping)

丢弃式原型法，是把原型系统作为用户和开发人员之间进行通信的媒介，并不打算把它作为实际系统运行。这与通常意义上的“模型”的概念相似，原始系统只是从外观、功能上“像”实际系统。开发这类原型系统的目的是为了对最终系统进行研究，使用户和开发人员借助这个系统进行交流，共同明确新系统的需求。比如，在设计一个仓库管理系统时，可以使原型系统的输入、输出数据限制在很小的范围内，可以没有各种输入提示、错误处理、文档说明等，但整个处理环节、流程和人机界面都具体地反映了出来。如果原型系统符合用户的需求，开发人员就可以把这些资料整理起来，作为初步的设计参考。

使用丢弃式原型法时，原型系统的开发过程可以作为传统的生命周期法的一个阶段，即需求定义阶段。这样，原型法就与传统的开发方法紧密地结合在一起了。这种形式的开发过程如图 3.3 所示。

由于原型系统在完成评审、开发之后就被扔掉，因而要求其开发费用低、速度快，通常需利用现有的软件工具及环境作为支持。

#### (2) 演化式 (Evolutionary Prototyping)

演化式原型法的开发思想与丢弃式完全相反，其思想为：用户的要求及系统的功能都无时不在地发生着变化，与其花大力气了解不清楚的东西，不如先按照基本需求开发出一个系统，让用户先使用起来，随时有问题随时修改。系统开始也许只能完成一项或几项任务，随着用户的使用及对系统的了解不断加深，原系统的一部分或几部分可能不再适应用户的要求，需重新设计、实施和安装。增加原系统功能在演化式原型法中极为频繁。

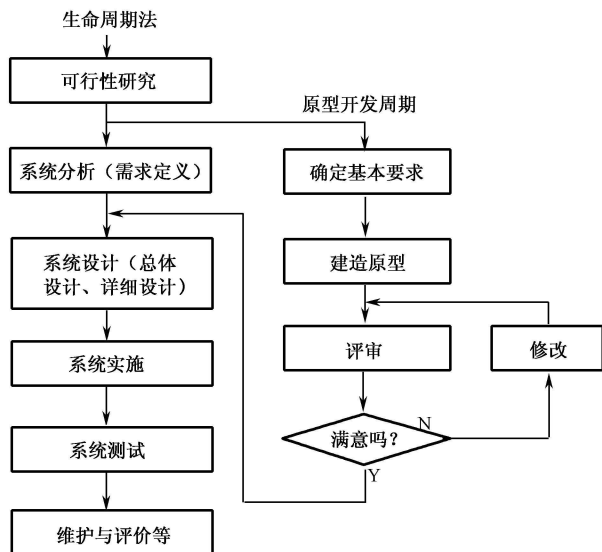


图 3.3 丢弃式原型法与生命周期法的结合过程

演化式原型法的开发过程一般由设计、实施和演化三个阶段组成，如图 3.4 所示。

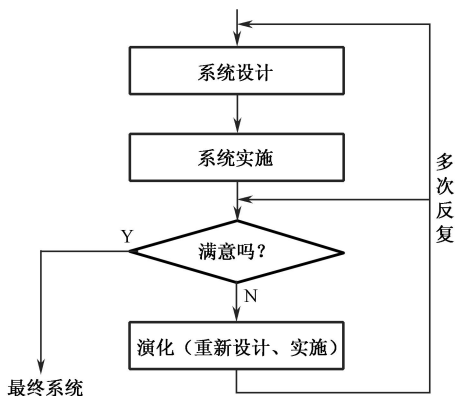


图 3.4 演化式原型法的开发过程

按照演化式原型法开发出来的系统即为最终系统，可立即投入正常运行。可以预料，由于在开发过程中反复进行修改，经常由用户对其评价，因此开发完成后的系统肯定会很好地满足用户的要求。

用演化式的开发方法在进行工程的实际实施时，要注意加强管理和控制，必须围

绕着系统的基本需求进行，否则会引起无休止的反复，使时间和费用都无法控制。

### （3）递增式（Incremental Prototyping）

递增式原型法与以上两种方法既有相同又有不同之处。根据这种方法，在开始时系统有一个总体框架，各功能单元的结构和功能也十分清楚，但还没有进行具体实现。这就是说，系统应完成什么功能、分为几个部分、各个部分应有几个模块，都已理解、掌握，且以后不需要作更大的变动，只是具体到每一个模块，还没有全部实现。但为了说明问题又都有一些演示数据说明这些模块的功能。这样，在以后的开发过程中，必须一个一个地完善这些模块。具体的设计可能是完全实现一个新的模块，也可能是用一个效率高的新模块代替一个旧模块。但所有这些工作都是基于一个前提：系统的组织结构不发生变化，模块的外部功能不发生变化。从某种角度考虑，这很类似于计算机工业中的插接策略（Plug-in Strategy）——要用到一个功能，就插上一个功能模块。

根据这种思想，递增式原型法的开发过程分为总体设计和反复进行的功能子单元实现两个阶段，如图 3.5 所示。

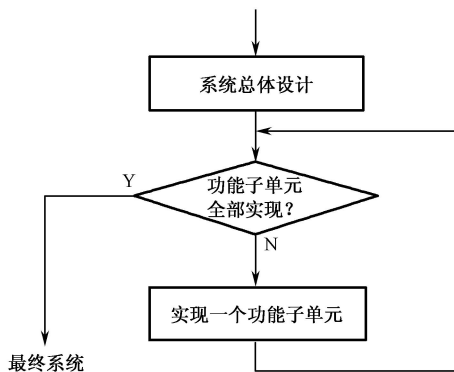


图 3.5 递增式原型法的开发过程

采用递增式原型法开发出的系统也是一个可实际运行的系统。

## 3.3 信息系统开发的过程模型

信息系统生命周期各个阶段的划分，是可粗可细的。实践中，各个阶段之间的关系也不可能仅是线性的、顺序的，而是带有反馈的迭代过程。这种过程通常用信息系统开发的过程模型来表示。

信息系统开发的过程模型给出了信息系统开发活动各阶段之间的关系。它是信息系统开发过程的概括，是信息系统工程的重要内容。

### 3.3.1 瀑布模型

瀑布模型（Waterfall Model），也称生命周期模型或线性顺序模型，是一种系统化的、线性的开发方法，由 W. Royce 于 1970 年首先提出。根据系统生命周期各个阶段的任务，瀑布模型从可行性研究开始，逐步进行阶段性变换，直至系统实施并最终使用维护，形成用户确认的系统产品。瀑布模型上一阶段的变换结果是下一阶段的输入，相邻两个阶段具有因果关系，紧密相联。一个阶段工作的失误将蔓延到以后的各个阶段。为了保证系统开发的正确性，每一阶段的任务完成后都必须对其阶段性产品进行评审，确认之后再转入下一阶段的工作。评审过程中发现错误和疏漏后，应该反馈到前面的有关阶段修正错误，弥补疏漏，然后再重复前面的工作，直至某一阶段通过评审后再进入下一阶段。这种形式的瀑布模型是带有反馈的瀑布模型。

瀑布模型主要包括开发和确认两个过程。

（1）开发过程是严格的下导式过程，各阶段间具有顺序性和依赖性，前一阶段的输出是后一阶段的输入，每个阶段工作的完成需要评审确认。

（2）确认过程是严格的追溯式过程，后一阶段出现了问题要通过前一阶段的重新确认来解决。问题发现得越晚，解决问题的难度就越大。

瀑布模型是系统开发中最基本的模型，它提供了系统开发的基本框架，有利于系统开发过程中人员的组织、管理，有利于系统开发方法和工具的研究，提高了系统开发的质量和效率。但是，从认识论角度可知，人的认识是一个多次反复的过程，即实践、认识、再实践、再认识，多次认识，多次飞跃，最后才能获得对客观世界较为正确的认识。瀑布模型恰恰没有反映这种认识过程的反复性。

因此，瀑布模型适合系统需求非常明确、设计方案确定及所有阶段都有较大把握的开发活动。

瀑布模型是最早也是应用最成功的工程范例。但是，在使用瀑布模型过程中有时会遇到如下一些问题。

（1）实际的项目很少按照该模型给出的顺序进行。虽然线性模型能够容许迭代，但却是间接的。在项目开发过程中的变化可能引起混乱。

（2）用户常常难以清楚地给出所有需求，而线性模型却要求如此，它还不能接受在许多项目的开始阶段就自然存在的不确定性。

(3) 用户必须要有耐心。可运行的系统一直要等到项目开发晚期才能得到。大的错误如果直到检查运行程序时才被发现，则其后果可能是灾难性的。

(4) 开发者常常被不必要地耽搁。在实际项目中，瀑布模型的线性特征可能会导致“阻塞状态”，某些项目组成员不得不等待组内其他成员先完成其依赖的任务。

### 3.3.2 原型模型

瀑布模型是一个严格的自顶向下模型，要求开发人员在初期就明确系统的需求，并对每一阶段都预先有较大的把握。原型模型（Prototyping Model）则恰好相反，它是开发人员根据用户提出的需求，借助一些软件开发工具或环境尽可能地快速构造一个实际系统的简化模型（原型），向用户展示待开发系统的全部或部分功能和性能，在征求用户对原型系统的过程中，进一步修改、完善、确认系统的需求并达到一致的理解的。

原型模型的特点是：首先，利用原型法技术能够快速实现系统的初步模型，供开发人员和用户进行交流，以便较准确地获得用户的需求；然后，采用逐步求精的方法使原型逐步完善，使得原型可以在新的层次上不断反复推进。

相对瀑布模型，原型模型更符合人类认识真理的过程和思维活动。但采用原型模型首先要有快速建立原型模型的软件工具和环境；其次原型模型适合于那些不能预先定义需求的开发活动。

原型模型从需求收集开始。首先，开发者和用户共同制定软件的总体目标，标识已知的需求，并规划进一步需要定义的需求。然后，进行“快速设计”，它主要集中在用户界面部分的表示（如输入方式和输出格式）。接着，建造原型，并由用户对原型进行评估，获得进一步精化的需求。之后，修改原型使其满足用户的要求，同时开发者对系统的功能有更好的理解。显然，这是迭代的过程。

理想上，原型可以作为标识系统需求的一种机制。如果建立了可运行原型，开发者就可以在此基础上利用已有的程序或使用工具（如报表生成器、窗口管理等）尽快生成工作程序。

原型可以作为“第一个系统”，但这可能是一种理想化的看法。用户和开发者确实都喜欢原型范型，用户能够感受到实际的系统，开发者能够很快地建造出一些东西。但由于如下的原因，原型仍存在问题。

(1) 用户看到的似乎是系统的工作版本，他们不知道原型只是拼凑起来的；不知道为了使原型能够很快工作，没有考虑软件的总体质量和长期的可维护性。当告知该产品必须重建才能使其达到高质量时，用户往往叫苦连天，会要求做“一些修改”，使

原型成为最终的工作产品。如此，系统开发管理常常就放松了。

(2) 开发者常常需要实现上的折中，以使原型能够尽快工作。一个不合适的操作系统或程序设计语言可能被采用，仅仅因为它是通用的和有名的；一个效率低的算法可能被使用，仅仅为了演示功能。经过一段时间之后，开发者可能对这些选择已经习惯了，忘记了它们不合适的所有原因。于是这些不理想的选择就成为了系统的组成部分。

(3) 虽然会出现问题，但原型仍是系统开发的一个有效范型。关键是如何定义一开始的游戏规则，即用户和开发者两方面必须达成一致：原型被建造仅是为了定义需求，之后就该被抛弃（或至少部分抛弃），实际的软件在充分考虑了质量和可维护性之后才能开发。

### 3.3.3 RAD 模型

快速应用开发（Rapid Application Development, RAD）是一个线性顺序的系统开发模型，强调较短的开发周期。RAD 模型是线性顺序模型的一个“高速”变种，通过使用基于构件的建造方法获得了快速开发。如果需求理解得很好，且约束了项目范围，则 RAD 过程可使一个开发组能够在很短的时间内创建出“功能完善的系统”。RAD 方法主要用于应用软件系统的开发，它包含如下几个开发阶段。

(1) 业务建模。业务活动中的信息流被模型化，以回答如下问题：什么信息驱动业务流程，生成什么信息，谁生成该信息，该信息流往何处，谁处理它。

(2) 数据建模。业务建模阶段定义的一部分信息流被精化，形成一组支持该业务所需的数据对象。数据建模阶段标识出每个对象的特征（称为属性），并定义这些对象间的关系。

(3) 处理建模。数据建模阶段定义的数据对象变换成为要完成一个业务功能所需的信息流。处理建模阶段创建处理描述，以便增加、修改、删除或获取某个数据对象。

(4) 应用生成。RAD 过程不是采用传统的第三代程序设计语言来创建软件的，而是复用已有的程序构件（如果可能的话）或是创建可复用的构件（如果需要的话）。在所有情况下，均使用自动化工具辅助软件建造。

(5) 测试及反复。因为 RAD 过程强调复用，所以许多程序构件已经是测试过的，这样减少了测试时间，但新构件必须测试，所有接口也必须测试。

RAD 模型如图 3.6 所示。

像所有其他过程一样，RAD 方法也有其缺陷。

(1) 对于大型的、但可伸缩的项目，RAD 需要足够的人力资源以创建足够的 RAD 组。

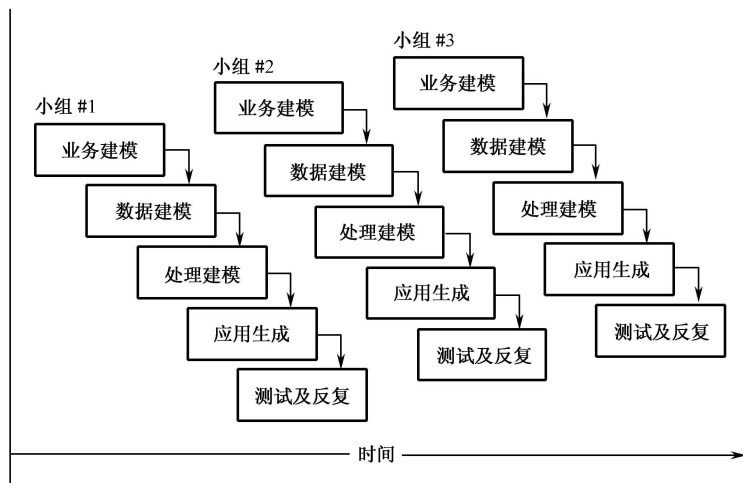


图 3.6 RAD 模型

(2) RAD 要求承担必要的快速活动的开发者和用户在一个很短的时间框架下完成一个系统。如果两方中的任何一方没有完成约定，则都会导致 RAD 项目的失败。

并非所有应用软件都适合使用 RAD。如果一个系统难以被适当地模块化，那么建造 RAD 所需的构件就会有问题；如果高性能是一个指标，且该指标必须通过调整接口使其适应系统构件才能赢得，则 RAD 方法就有可能失败了；RAD 不适合技术风险很高的情况，当一个新应用要采用很多新技术，或当新软件要求与已有计算机程序有高可互操作性时，这种情况就会发生。

### 3.3.4 增量模型

增量模型融合了线性顺序模型的基本成分（重复地应用）和原型模型的迭代特征。如图 3.7 所示，增量模型采用随着日程时间的进展而交错的线性序列。每一个线性序列产生软件的一个可发布的“增量”。

当使用增量模型时，第一个增量往往是核心的产品，即实现了基本需求，但很多补充的特性（其中一些是已知的，另外一些是未知的）还没有发布。核心产品交用户使用，使用和评估的结果是下一个增量的开发计划。该计划包括对核心产品的修改，使其能更好地满足用户的需要，并发布一些新增的特点和功能。这个过程在每一个增量发布后不断重复，直到产生最终的完善产品。



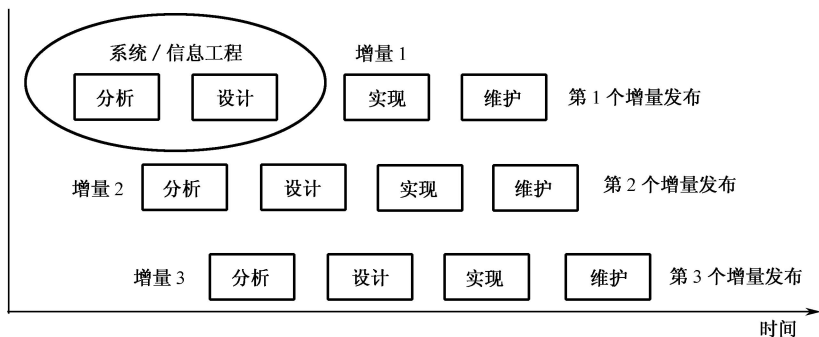


图 3.7 增量模型

增量模型像原型模型和其他演化方法一样，具有迭代的特征。但与原型模型不一样，增量模型强调每一个增量均发布一个可操作产品。早期的增量是最终产品的“可拆卸”版本，但它们确实提供了给用户服务的功能，并且提供了给用户评价的平台。

### 3.3.5 螺旋模型

螺旋模型（Spiral Model）是 B. Boehm 于 1988 年提出的。它是瀑布模型与原型模型的结合，不仅体现了两个模型的优点，而且还增加了新的成分——风险分析。螺旋模型的结构如图 3.8 所示。它由四个部分组成：

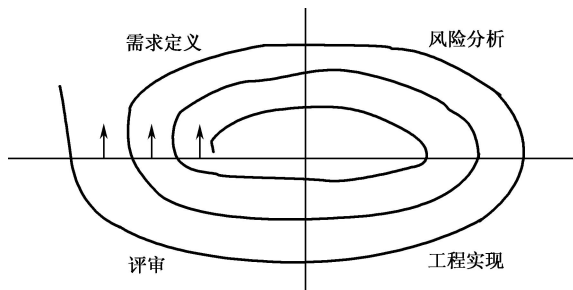


图 3.8 螺旋模型

- (1) 需求定义；
- (2) 风险分析；
- (3) 工程实现；
- (4) 评审。

螺旋模型是由上述四个部分组成的迭代模型。系统开发过程每迭代一次，螺旋线

就增加一周，系统开发又前进一个层次，系统又生成一个新版本，而系统开发的时间和成本又有了新的投入。在大多数场合，系统开发过程是沿螺旋线的路径连续进行的，最后总能得到一个用户满意的系统。理论上，迭代过程可以无休止地进行下去，是一个无限过程，但在实践中，迭代结果必须尽快收敛到用户允许的或可接受的目标范围内。只有降低迭代次数、减少每次迭代的工作量，才能降低系统开发的时间和成本。反之，如果迭代过程收敛得很慢，每迭代一次的工作量很大，则由于时间和成本上的开销太大，用户无法支持，系统开发往往不得不中途夭折。

螺旋模型的每一周期都包括需求定义、风险分析、工程实现和评审四个阶段。这是对典型生命周期模型的发展。它不仅保留了生命周期模型中系统地、按阶段逐步地进行系统开发和“边开发，边评审”的风格，而且还引入了风险分析，并把制作原型作为风险分析的主要措施。用户始终关心、参与开发工作并对阶段性的产品提出评审意见，这对保证产品的质量是十分有利的。

螺旋模型是支持大型系统开发并具有广泛应用前景的模型。

### 3.3.6 构件组装模型

对象技术为系统开发的基于构件的过程模型提供了技术框架。面向对象方法强调了类的创建，类封装了数据和用于操纵该数据的算法。如果经过合适的设计和实现，则面向对象的类可以在不同的应用及基于计算机的系统结构中复用。

构件组装模型融合了螺旋模型的许多特性。它本质上是演化的，支持系统开发的迭代方法。但是，构件组装模型是利用预先包装好的软件构件来构造应用程序的。

开发活动从候选构件的标识开始。这一步通过检查被应用程序操纵的数据及用于实现该操作的算法来完成。相关的数据和算法封装成一个构件。

以前的系统开发项目中创建的构件被存储在一个构件库中。一旦标识出候选构件，就可以搜索该构件库，确认这些构件是否存在。如果已经存在，就从库中提取出来复用。如果不存在，则采用面向对象的方法开发它。之后就可以利用从库中提取出来的构件或者为了满足应用程序的特定要求而建造的新构件，来构造待开发应用程序的第一个迭代。过程流程后又回到螺旋，并通过随后的工程活动最终再进入构件组装迭代。

### 3.3.7 组合模型

前面介绍的开发模型各有各的优、缺点。在工程实践中，经常把几种模型组合在一起，配套使用，形成组合模型（Combining Model）。

组合模型通常是以一种模型为主，嵌入另外一种或几种模型。例如，在生命周期

模型中，为了帮助用户和系统开发人员尽快确定系统需求，可以在系统分析阶段嵌入原型模型；在螺旋模型中，为了有利于风险分析和生成阶段性的代码，可以嵌入原型模型和生命周期模型。

### 3.3.8 形式化方法模型

形式化方法模型包含了一组活动，它们带来了系统应用软件用数学说明描述的方法。形式化方法使得开发人员能够通过采用一个严格的、数学的表示体系来说明、开发和验证基于计算机的系统。

当在开发中使用形式化方法时，它们提供了一种机制，能够消除使用其他开发模型难以克服的问题。二义性、不完整性和不一致性能被更容易地发现和纠正——不是通过专门的复审，而是通过数学分析。当在设计中使用形式化方法时，它们能作为程序验证的基础，从而使得开发人员能够发现和纠正在其他情况下发现不了的错误。

形式化方法模型虽然不是主流的方法，但可以产生正确的软件。不过，它存在如下一些缺陷：

- (1) 形式化模型的开发目前还很费时和昂贵；
- (2) 开发人员需要具有使用形式化方法所需的背景知识；
- (3) 难以使用该模型作为与对其一无所知的用户进行通信的机制。

## 3.4 信息系统开发方法学

信息系统开发方法学（Information Systems Development Methodologies）是研究信息系统开发规律的学科分支，其研究领域包括：

- (1) 信息系统开发的认知体系；
- (2) 信息资源的战略规划；
- (3) 信息系统开发策略；
- (4) 信息系统分析与设计的一般理论和方法；
- (5) 信息系统集成学；
- (6) 系统开发环境和技术；
- (7) 其他相关技术和方法。

迄今为止，在信息系统开发方法学领域尚未形成一套公认的最合理、科学的理论，以及由这些理论所支持的具体工具和方法。

### 3.4.1 系统开发认知体系

信息系统开发认知体系所要研究的是人们在信息系统开发过程中对其开发对象的认识方法，以及信息系统开发的规律及方法，以指导人们进行信息系统研制的指导思想、过程步骤及工具，为建立和认识客观事物及各种开发理论、方法奠定统一的基础和认知框架。

认识规律的研究主要是指在信息系统开发过程中人对客观事物（即将要被处理的对象）认识规律的研究。信息系统开发认知体系的研究目前主要有如下几个方面：

- （1）管理过程和方法的规范化、科学化研究；
- （2）管理过程和方法的形式化表达方法研究（或称从客观世界向机器处理过渡的方法研究）；
- （3）对被处理对象存在问题的认识方法和识别过程研究；
- （4）系统开发过程的指导思想和结构方法研究；
- （5）人们对于信息系统开发规律和认识规律的研究；
- （6）人们在从事与信息系统开发有关的活动中思维规律的研究；
- （7）系统开发认知方法及其知识层次的研究；
- （8）其他方面。

### 3.4.2 系统开发方法学

方法是为获取某一对象（如客观世界的某一结构等）而采用的组织人们思维活动的过程，以及实现这个过程所必需的步骤和途径。方法学是研究方法的科学。信息系统开发方法学是一门具体学科（信息系统开发）的方法学，其基本任务是研究信息系统开发的规律及相应的技术和工具，从认识论、方法论、系统论的角度研究出一套符合现阶段人们认识程度的系统开发原则、方法和工具，以指导开发实现的全过程。

系统开发方法学的研究是与认知体系的研究和工具方法的研究密切相关的。系统开发方法学的研究内容包括：

- （1）在较高层次上分析和总结以往的经验，研究信息系统开发的一般规律，建立具有一般意义（普遍适用）的系统开发指导思想的基本原则；
- （2）从系统工程的角度，为分析人员（或称信息系统的建造者）提供一个协调局部与整体利益的思维方法及具体的分析、设计原则；
- （3）围绕已建立的各种开发方法、指导思想的原则，建立相应的实施步骤；
- （4）研制一整套与系统开发思想相对应的，适合于各实施步骤的描述和开发工具；

- (5) 信息系统开发中的组织、实施方法；
- (6) 系统开发成功的关键因素、必要条件及促使系统开发成功的组织运行机制；
- (7) 其他。

### 3.4.3 系统开发策略与资源规划

信息系统的开发策略和信息资源规划，是指在一个系统具体地被开发前所进行的整体开发战略和资源规划。信息系统开发策略与资源规划所要研究的主要内容如下所述。

- (1) 组织的性质、特点、目标、地位及组织的发展战略规划。
- (2) 信息系统的任务、目标、作用、地位及所期望对组织管理的影响。
- (3) 现有资源分析。它包括对现有信息资源、信息处理能力、技术基础、环境条件、资金设备等资料的分析。
- (4) 组织的管理现状和需求分析。它包括：组织的管理体制、现状、水平；基础数据管理状况；组织今后战略发展可能对其现有管理模式所产生的影响；组织各部门对信息和信息系统的需求状况和需求层次等资料的分析。
- (5) 信息系统开发方法、投资方案，以及开发过程的实施规划。
- (6) 其他。

### 3.4.4 信息系统开发方法的规范化研究

信息系统开发方法的规范化研究是系统开发方法学研究的内容之一。在信息系统开发过程中采用工厂化、系统化的方法，其目的是综合考虑各方面的因素，协调力量，统一信息系统的开发口径，便于横向沟通等，以利于加速系统开发的水平和速度。近年来方法学领域中各种不同方法、工具的研究推动了方法学研究的发展，同时也带来了一些问题，引起了某些混乱。为解决这一问题，各国在开展方法学研究的同时，又相继展开了开发规范的研究。我国于20世纪80年代后期进行了这方面的研究，如1986年国家计委和国家标准局批准的“国家经济信息系统设计与应用标准化规范”，1989年12月国家体改委经济研究所设计的“企业管理信息系统开发规范”，等等，都是我国学者在这方面努力的结果。

## 习 题

- 1. 查阅其他参考书籍中关于信息系统生命周期的划分，具体分析其中不同之处，给出自己的理解。
- 2. 根据自己的理解，针对不同开发方法总结归纳出各自的优劣。



## 第4章 信息系统的战略规划与可行性研究

信息系统的建设是一项复杂的系统工程，它受到许多技术和非技术因素的影响，如社会政治、经济、文化等方面；同时，信息技术和信息系统在我国的应用的思维方式、组织的管理思想等方面还停留在传统模式上，这使得信息系统实施和应用受到较大的阻力。因此，针对组织的具体情况来规划和实施信息系统成为组织者和研究工作者关注的焦点。

战略规划通常指关于一个组织的发展方向、环境条件、长期目标、重大政策与策略等方面的规划。一个组织不仅在最高层有战略规划，而且在中层和基层也有战略规划。每层的战略规划都应符合上层战略规划的约束。任何组织的战略规划都在动态中发展，而且在不同时期，可能需要根据环境条件和政策策略进行调整。

电子计算机和现代通信技术的结合使信息资源的开发和利用摆脱了传统的迟缓与分散的方式，逐步走上了高效率、专业化、多样化的开发利用阶段，信息已成为生产力中最主要的因素之一，成为社会发展的战略资源。通过信息资源的开发利用来加速提高人的素质，加快科技文化的进步，促进物质和能源的高效利用，使农业、工业、服务业取得更高的效率和效益，是国民经济信息化的本质所在，是信息系统开发的最终目标。

本章主要讲述信息系统战略规划的概念、目标、作用、内容与组织；制定信息系统战略规划的步骤；战略规划的常用方法及信息工程与战略数据规划。

### 4.1 信息系统战略规划的概念、目标与组织

下面介绍信息系统战略规划的概念、目标、作用、内容和组织。

#### 4.1.1 信息系统战略规划的概念与层次

##### 1. 战略规划的概念

美国学者詹姆斯·马丁（James Martin）在《战略数据规划方法学》和《数据库环境的管理》等书中结合信息系统开发实例，完整、系统地论述了信息系统的开发策略和方法学。

一个组织的战略规划应当是有效的，所谓有效包括两个方面：一是正确性，即战略规划中的战略是正确的，也就是说应做到组织资源的良好利用和与环境的良好匹配；二是可行性，即战略规划中的战略适合于该组织的管理过程，也就是与组织活动的良好匹配。因此，一个有效的战略规划具有以下特点。

(1) 可执行性：战略明确、容易理解、可操作，能真正起到对管理工作的指导作用。

(2) 可落实性：战略规划要求逐级落实。上级部门制定的战略一般应以方向和约束的形式传达至下级，下级接受任务并将之细化，然后再以同样的方式向下传达，整个战略规划通过不断细化直至落实到不可分解为止。

(3) 可变更性：战略规划应当进行周期性地复核和评审，这就要求战略规划有较强的灵活性，容易适应变更的需要。

## 2. 信息系统战略规划的层次

信息系统战略规划（Information System Strategic Planning, ISSP）可划分为国家级、行业级和企业级三个层次。

(1) 国家级：为了实现一个国家的国民经济信息资源的有效开发和利用所进行的总体规划是国家级信息系统战略规划。

国家级的国民经济信息化总体规划主要包括以下六个方面：基础理论，信息技术，推进国民经济信息化的管理体制和法律法规，信息产业，国家信息基础设施，国民经济各领域的信息化。它们之间的关系如图 4.1 所示。

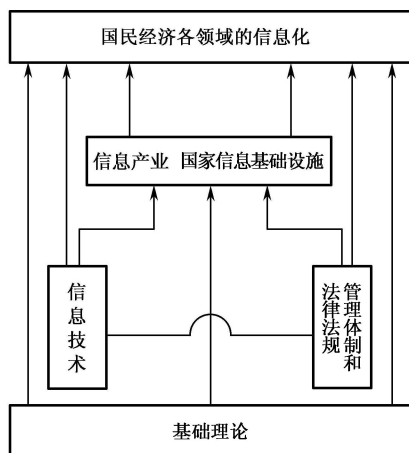


图 4.1 国民经济信息化总体规划构成示意图



推进国民经济各领域的信息化是国民经济信息化的主要目的；信息产业和国家信息基础设施是各领域信息化的基础，也是国民经济信息化的支柱；信息技术是信息产业发展和国家信息基础建设的基础；国民经济信息化的管理机制和法律法规是技术和基础设施、产业应用健康快速发展的必然要求；基础理论（包括信息技术的基础理论、国民经济信息化发展规律和发展战略、信息产业和国家信息基础设施建设的规律、各领域应用的发展规律）则是制定科学的、适合中国国情的国民经济信息总体规划的基础，也是实施国民经济信息化的基础。

（2）行业级：行业级信息化总体规划则要在国民经济信息化总体规划的指导下，从以下五个方面来进行：行业级信息技术，行业内部管理体制和法律法规，行业级信息基础设施，行业内部信息资源规划与标准化，行业内部各组织的信息化。

（3）企业级：企业或组织内部的总体规划则要在国家级、行业级总体规划的指导下，从以下三个层次上进行：战略的业务规划，战略的信息技术规划（包括企业或组织内部信息技术规划、信息基础设施规划、数据管理策略、应用开发策略、分布处理策略等），战略的数据规划。如图 4.2 所示。

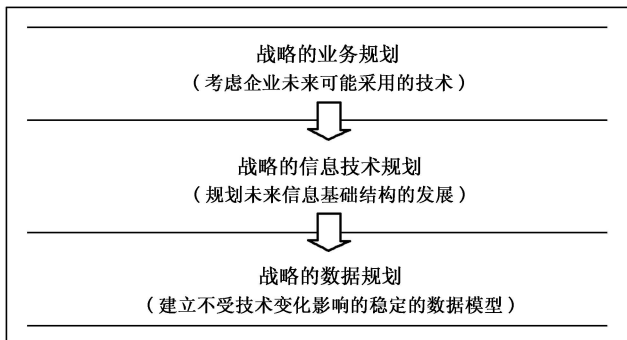


图 4.2 企业总体规划的三个层次

所有企业或组织都应该有战略的业务规划。战略的业务规划描述组织的基本目标、发展战略和组织指标。现在各项技术的发展正在改变组织的各个方面，在某些情况下正在改变组织内的业务类型，因此，战略的业务规划不能与各项技术发展无关。技术在改变产品，改变制造方法，改变服务，改变信息流通方式，改变决策的制定方式，并由此影响管理机构的变化。战略的信息技术规划要对组织内部信息基础设施的建设加以规划，只有这样才能使分布式系统、数据库系统得到健康的发展。战略的数据规划则要对组织内部中的数据实体及它们的属性进行规划。通常，组织内部在其目标不变的前提下，数据模型相对稳定，处理可能在不断地变化，因此，可以通过总体数据

规划建立稳定的数据模型，这个模型将是富有生命力的，同时可以依据这个模型建立组织内的公用数据库并开展各项应用项目的开发。

### 3. 信息系统战略规划

信息系统战略规划是关于信息系统长远发展的规划。它既可以看成是企业战略规划的一个重要组成部分，也可以看成是企业战略规划下的一个专门性规划。

信息系统战略规划主要解决如下四个问题：如何保证信息系统规划与它所服务的组织总体战略上的一致？怎样为该组织设计出一个信息系统总体结构，并在此基础上设置、开发应用系统？对相互竞争的应用系统，应如何拟定优先开发计划和运营资源的分配计划？面对前三个阶段的工作，应怎样选择并应用行之有效的设计方法论？

信息系统的战略规划是信息系统生命周期中的第一个阶段，也是系统开发过程的第一步，其质量直接影响着系统开发的成败。正是由于信息系统是一项耗资巨大、技术复杂、开发周期长的系统工程，所以需要一个高层的战略规划，也就是以整个系统为分析对象，从战略上把握系统的目标和功能的框架。

在现代社会中，信息已成为企业的生命线，信息资源是企业的一项重要财富，信息管理是企业管理的重要组成部分，信息系统的运行与企业的运营方式息息相关，所以不仅要在资源上、经费上、时间上给予充分考虑，而且要在观念上给予高度重视，做出全方位的规划。许多事实还证明，信息系统战略规划还可以直接为企业带来积极影响，如更准确地识别出哪些是实现企业目标所必须完成的任务；发现过去可能没有发现的潜在问题，为企业更合理地安排各种业务活动提供依据等。

信息系统战略规划有狭义和广义两个概念。广义的信息系统战略规划是指信息系统的整个建设计划，既包括战略计划，也包括信息需求分析和资源分配。狭义的信息系统战略规划则不包括后面分析的内容。

一般情况下，如果将信息系统战略规划看成是企业战略规划下的一个专门性规划，则它将是在制定企业战略之后配合其结果和要求来制定的。另一种情况则将信息系统战略规划看成是企业战略规划的一个组成部分，在制定企业战略规划中的生产规划、市场规划等的同时制定信息系统战略规划。由于信息的规划涉及生产、市场等多个部门的规划，因此要强调信息系统战略规划与企业战略规划整体的协调。总之，不论信息系统战略规划是作为企业战略规划的一部分还是一个专门性规划，都应与企业战略规划有机地配合。正如一些信息系统规划专家所指出的：如何使一个企业中的信息系统发展战略与企业发展战略保持一致，是信息系统战略规划工作的核心问题之一。

## 4. 战略规划与信息系统

当一个企业制定和调整企业战略规划时，它完全可以借助已有的信息系统提供支持。因为信息系统能提供许多连续的、规范化的、包括企业内外两方面的、原始的或经过分析的信息来支持企业战略规划制定的全过程。

首先，在战略形成过程中，信息系统可以支持环境分析，支持数据收集、输入与加工，支持模型构造、报告与查询。例如，构造一个模型来评价未来五年中企业资源的供应情况。其次，在战略规划决策中，信息系统又可以支持设备选择、新产品开发、合同投标、市场扩展、财务计划等有关决策。例如，面对许多新产品开发方案，选择一个在利润和风险方面具有最佳结果的方案。信息系统还能帮助规划短期活动，如资源利用、产品选择、市场研究和销售计划等。

### 4.1.2 信息系统战略规划的目标、作用、内容与组织

#### 1. 战略规划的目标

信息系统战略规划的目标是制定与组织发展战略目标相一致的信息系统发展战略目标。

目前，在信息系统战略规划工作中，存在着两种性质截然不同的发展战略：一种战略是希望通过更多更好的硬件和软件来增加系统的数据处理能力；另一种战略则是强调建立更好的组织模式，目的是给计划和控制提供良好的管理信息。不论哪一种战略，都必须根据以前的情况来预测战略规划执行期间的技术和管理上的进展，而且也要考虑将来的组织结构、产品情况和业务系统，更重要的是要确保所制定的信息系统战略规划的目标与组织的战略规划的目标相一致。

#### 2. 战略规划的作用

信息系统战略规划工作的好坏是信息系统成败的关键。一个有效的信息系统战略规划可以使信息系统和用户建立较好的关系，可以做到信息资源的合理分配和利用，从而节省信息系统的投资。一个有效的信息系统战略规划可以促进信息系统应用的深化，为企业带来更多的经济效益。一个有效的信息系统战略规划还可以作为一个标准，考核信息系统开发人员的工作，明确他们的努力方向。一个信息系统战略规划的制定过程本身也是迫使企业领导回顾过去的工作，发现可改进的地方的过程。只有进行信息系统战略规划，才可以保证信息系统中信息的一致性，避免信息系统成为沙滩上的房屋。

### 3. 战略规划的内容

信息系统战略规划一般既包含三年或更长的长期计划，也包含一年的短期计划。长期计划部分指明了总的发展方向，而短期计划部分则为作业和资金工作的具体责任提供依据。一般说来，整个战略规划包括四项主要内容。

（1）信息系统的目标、约束与结构：战略规划包括企业的战略目标、外部环境、内部环境、内部约束条件、信息系统的总目标、计划和信息系统的总体结构等。其中信息系统的总目标为信息系统的发展方向提供准则；计划是对完成工作的具体衡量标准；信息系统的总体结构规定了信息的主要类型及主要的子系统，为系统开发提供了框架。

（2）当前的能力状况：战略规划包括硬件情况、通用软件情况、应用系统及人员情况、硬件与软件人员及费用的使用情况、项目进展状况及评价等。

（3）对影响计划的信息技术发展的预测：信息系统战略规划自然要受到当前和未来信息技术发展的影响。计算机硬件技术、网络技术、数据库技术及办公自动化技术等的影响应能够准确觉察并在战略规划中有所反映。对软件的可用性、方法论的变化、周围环境的发展及它们对信息系统产生的影响也应该在所考虑的因素之中。这些是信息系统有较强生命力的保证。

（4）近期计划：在战略规划适用的几年中，应对即将到来的一段时期（如一年）做出相当具体的安排，主要应包括硬件设备的采购时间表、应用项目开发时间表、软件维护与转换工作时间表、人力资源的需求及人员培训时间安排、财务资金需求等。

信息系统的战略规划需要不断地修改。人员的变化、技术的变革、组织自身的变化都可能影响到整个规划，甚至一种新的硬件或软件的推出也能影响到整个规划。除此之外，修改规划的原因还可能来自信息系统之外的事物，如财务限制、政府的规章制度、竞争对手采取的行动等。

### 4. 战略规划的组织

（1）规划领导小组：战略规划既要考虑规划内容，也要考虑规划中所提出的方方面面之间的相互关联。为了实现规划目标，首先必须组织一支在最高层领导支持下的强有力的规划队伍，通常称为信息系统规划领导小组。这个小组要在企业最高层管理者的直接领导之下，由一名负责全面规划工作的信息资源规划负责人和企业中有关部门的主要负责人组成，并通过一批用户分析员与广大的最终用户相联系。其中有关部门的主要负责人应包括数据处理负责人、系统分析负责人、财务负责人、各业务经理等。信息资源规划负责人应掌握一套成熟的科学规划方法，这样的负责人最好出自企业内的最高层管理人员。信息系统的最终用户是指那些直接使用应用系统的各层管理人员，

包括高层管理人员、中层管理人员和基层管理人员。这些人员中要抽出一部分人在战略规划期间代表所在的部门参加规划工作，这也就是前面所说的用户分析员。用户分析员既是规划工作的具体参加者，又是规划领导小组与广大管理人员的联系者，将来还是系统的最终用户。

强调规划领导小组要由企业的主要负责人及企业中各个部门的主要负责人组成，一方面是为了能够从组织的最高管理层次出发来保证系统规划的重要性及权威性，另一方面也能够协调组织内部各部门对信息系统的不同要求，统一使用有限的资源，以保证各部门对信息系统开发工作的有效支持。在信息系统的规划完成以后，规划领导小组实际上就转成了信息系统建设领导小组，由它来决定开发哪些信息系统的应用项目，并组织有关人员完成系统规划所提出的要求。在信息系统技术不断深入到社会各领域的今天，企业中的信息系统领导小组应该成为一种长期性的组织。

(2) 人员培训：一个企业准备进行战略规划，意味着要采用一套科学的方法进行信息系统的基础建设。这套方法对大多数参加者来说是陌生的，因而必须通过适当的培训使他们掌握这套方法。可以说，能否使参加规划的人掌握科学的方法，是战略规划工作成功的关键因素之一。培训包括最高层管理人员的培训、用户分析员的培训及规划领导小组其他成员的培训。通过培训使他们学会识别企业过程、分析业务活动等项工作。

(3) 时间规定：自顶向下的规划一般应在半年内完成。根据对一些企业实际规划情况的考察可知，只要有明确的方向并采取切实可行的规划方法，大多数战略规划工作都可以在这个时间内完成。如果缺少切实可行的规划方法，规划过程不加严格管理，则自顶向下的规划工作可能被拖得很长，以致于有时会丧失信誉或被迫放弃。

## 4.2 信息系统战略规划的步骤

### 4.2.1 诺兰的阶段模型

在一些行业或企业，信息系统的建设刚刚起步。而在另一些行业或企业，信息系统的建设已经趋于成熟，诺兰的阶段模型反映了信息系统的发展阶段并使信息系统的各种特性与系统生长的不同阶段对应起来，从而成为信息系统规划工作的框架。根据这个模型，只要一个信息系统存在某些特性，便知其处在哪一阶段。其基本思想是，一个企业的信息系统在能够转入下一阶段之前，必须首先经过系统生长的前几个阶段。因此，如果能够诊断出一个企业目前所处的成长阶段，就能够对它的战略规划提出一系列的限制条件和做出有针对性的规划方案。

诺兰（Nolan）在 1973 年首次提出的信息系统发展阶段理论仅确定了信息系统生长的四个不同阶段，到 1980 年，诺兰又把上述模型扩展成了六个阶段，如图 4.3 所示。

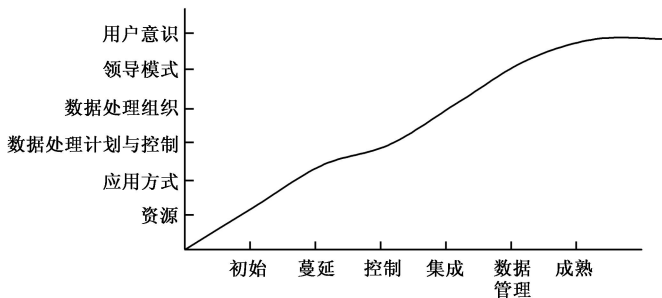


图 4.3 诺兰的阶段模型

图 4.3 中的水平轴列出了六个发展阶段，垂直轴列出了增长要素，曲线表示六个阶段的信息系统预算。显然，曲线基本上是 S 形的，即在第一、二阶段预算上升很快，在第三阶段较为平缓，在第四阶段又逐步上升，在第五、六阶段又变得平缓。

（1）第一阶段：初始阶段。企业购置第一台用于管理的计算机，标明信息系统开发初始阶段的开始。在这一阶段，各级管理人员对信息系统从不认识到有点认识，支持、组织开发出了一两个简单的应用系统。初始阶段的计算机一般是在会计、统计等部门。这些简单的应用系统的运行所产生的效益和效率使得人们对信息系统的认识大大提高，于是逐渐进入了蔓延阶段。

（2）第二阶段：蔓延阶段。随着计算机应用显现效果，信息系统从最初的一些部门向各个部门扩散。这一阶段是数据处理发展最快的一个阶段，用户感到计算机在事务处理上的好处，计算机利用率开始不断地提高，各部门都开发了大量应用程序，但这时由于缺乏综合系统开发，所以出现了信息冗余、代码不一致、信息难以共享等混乱局面。在 20 世纪 60 年代，美国多数公司经历了这个阶段，当时由于无控制的技术刺激和松弛的管理，计算机应用猛增，但只有一部分收到了良好的效益。

（3）第三阶段：控制阶段。由于广大管理人员都认识到了计算机信息系统的优越性，纷纷购置设备，开发支持自身管理的信息系统，所以使得硬件、软件投资和开发费用急剧增长，增长到一定程度便会受到控制，即进入控制阶段。这个阶段除了各项投资费用受到控制外，还要求完善各个子系统的功能以提高现有计算机应用的效益，其发展速度与前两个阶段相比要缓慢得多。管理部门了解到他们的计算机系统的投资超出了控制，即预算每年以 30%~40% 的比例增长，而投资的回收却不理想。同时随着应用经验逐渐丰富，应用项目不断积累，客观上也要求加强组织协调，于是就出现



了由企业领导和职能部门负责人参加的领导小组，对整个企业的系统建设进行统筹规划，特别是利用数据库技术解决数据共享问题。这时，严格的控制阶段便代替了蔓延阶段。诺兰认为，第三阶段将是实现从以计算机管理为主转向以数据管理为主的阶级关键，一般发展较慢。

(4) 第四阶段：集成阶段。由于发现分散开发的系统不能互通、信息不能共享等一系列问题，于是产生了从全局出发，建立一个支持全企业的信息系统的需求，即进入了集成阶段。在集成阶段，信息系统的开发首先考虑总体，面向数据库建立稳定的全局数据模型，基于稳定的全局数据模型实现各子系统的功能需求，进而发挥信息“黏合剂”和“倍增剂”的作用。这种开发支持全局信息系统的需求势必带来各项投资费用的增长，但开发速度加快了。

(5) 第五阶段：数据管理阶段。诺兰认为，在集成阶段之后才会真正进入数据管理阶段。这时，数据真正成为企业的重要资源。鉴于美国在 20 世纪 80 年代时多数企业还处在第四阶段，所以诺兰对第五阶段还无法给出详细的描述。

(6) 第六阶段：成熟阶段。一般认为，信息系统的成熟表明它可以满足企业的各管理层次的要求，从操作层的事务处理到中间管理层的控制管理（信息系统），到支持高级管理层的决策支持，真正实现了信息资源的管理。

图 4.3 中还显示了六种增长要素。第一是资源，主要指计算机的硬、软件资源。第二是应用方式，如批处理方式和联机方式。第三是数据处理计划与控制，从开始的随机的、短期的计划到长期的、战略的计划。第四是数据处理组织，确切地说是信息系统功能在组织中所占的地位。在早期，信息处理功能常归属于财务部门，计算机被看作是与计算器一样的附属品，到第三、四阶段后，信息系统才发展成独立的活动部分。第五是领导模式。在第一、二阶段，技术领导是主要的，随着用户和上层管理人员越来越了解信息系统，在第五、六阶段，上层管理部门开始与信息系统管理部门一起决定发展战略。第六是信息系统用户意识的改变，即从操作管理级的用户发展到中层和上层管理级。

诺兰模型是对计算机信息系统发展历程的总结，诺兰曲线是一种波浪式的发展过程，反映了一定的发展规律，跳跃某个或某几个阶段是不大可能的，但是随着人们对信息系统认识的提高，可以压缩某些阶段的时间，特别是蔓延阶段的时间。

诺兰的阶段理论既可以用于诊断当前处在哪个生长阶段、向什么方向前进、怎样管理对研制最有效，也可以用于对各种变动的安排，进而以一种可行方式转至下一生长阶段。虽然系统生长现象是连续的，但各阶段则是离散的。在制定战略规划的过程中，根据各阶段之间的转换和各种特性的逐渐出现，运用诺兰模型辅助规划的制定，并将它作为信息系统规划指南是十分有益的。

4.2.2 信息系统战略规划的三阶段模型

目前已有许多方法用于信息系统的战略规划工作，各种方法在规划中所起的作用和地位是不同的。由 B. Bowman、G. B. Davis 等研究的信息系统战略规划工作的三阶段模型，阐明了广义战略规划的制定活动及各活动的顺序与可选用的技术和方法，如图 4.4 所示。

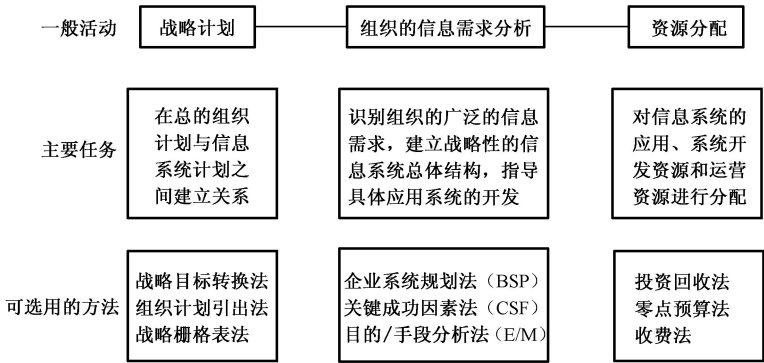


图 4.4 信息系统战略规划的三阶段模型

该模型有助于人们了解规划问题的本质并选择适当的规划阶段，可减少规划方法使用不当造成的混乱，对信息系统的规划具有指导意义。

4.2.3 信息系统战略规划的步骤

制定信息系统战略规划的具体步骤如下。

- (1) 确定规划性质：检查企业的战略规划，确定信息系统战略规划的年限和规划方法。
- (2) 收集相关信息：收集来自企业内部和环境中的与战略规划有关的各种信息。
- (3) 进行战略分析：对信息系统的战略目标、开发方法、功能结构、计划活动、信息部门情况、财务状况、所承担的风险程度和政策等多方面进行分析。
- (4) 定义约束条件：根据财务资源、人力资源、信息设备资源等方面的限制，定义信息系统的约束条件和政策。
- (5) 明确战略目标：根据分析结果与约束条件，确定信息系统的战略目标，也就是在规划结束时，信息系统应具有怎样的能力，包括服务的范围、质量等多方面。
- (6) 提出未来略图：勾画出未来的信息系统的框图，产生子系统划分表等。
- (7) 选择开发方案：对信息系统进行分析，根据资源的限制选择一些适宜的项目



优先开发，制定出总体开发顺序。

(8) 提出实施进度：在确定每个项目的优先权后，估计项目成本、人员要求等，以此作为整个时期的任务、成本与进度表。

(9) 通过战略规划：撰写战略规划书，书写过程中不断征求用户、信息系统工作者的意见。战略规划经企业领导批准后生效，并将它合并到企业战略规划中。

### 4.3 信息系统战略规划的常用方法

#### 1. 组织计划引出法

如果企业有自身的目标和战略的总体规划，那么就可以从该规划中引出信息系统的目标和战略。一般情况下，应首先对企业战略规划中的每一目标和战略加以分析，然后根据分析的结果和对信息系统的要求进行组织并写入信息系统的规划中。例如，一个企业战略规划中的某项目标是，2009 年 12 月 31 日前在全厂实施全面质量管理体系，则导出的信息系统战略规划的目标可以是，在 2009 年 12 月 31 日前产生适合于全面质量管理体系的质量控制报告，为此在 2009 年 6 月 30 日前为全面质量管理体系的推行而设计质量控制数据库的管理规范。该企业战略规划中的战略是为新产品建立全面质量管理控制规程，由此导出的信息系统战略规划中的战略是建立新产品的全面质量管理控制数据库系统。

#### 2. 战略栅格表法

战略栅格表法是一种了解企业中信息系统作用的诊断工具，该方法利用栅格表，依据现行应用项目和预计将开发的应用项目的战略影响，确定出四种不同类型的信息系统规划条件，即战略、转换、工厂和辅助，如图 4.5 所示。

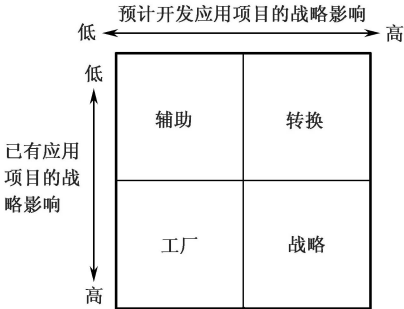


图 4.5 战略栅格表

栅格表中的每一方格确定了企业信息活动的位置。通过对当前应用项目和将开发应用项目可能产生的影响的分析，可达到诊断当前状态和调整战略方向的作用。例如，若分析结果表明企业的信息系统处于“战略”位置，则说明信息系统的工作对当前的竞争策略和企业未来的战略方向都是至关重要的。若处于“辅助”位置，则表明信息系统的应用对企业的各项活动是一种辅助。若处于“工厂”位置，则表示信息系统的应用对成功地执行那些严格规定和广为接受的活动极为重要，但信息系统还不是未来战略活动的组成部分。若处于“转换”位置，则是一种从“辅助”到“战略”的转变状态。战略栅格表中的每一位置说明了上层管理人员需要介入的程度，以及信息系统规划与企业的战略规划之间的关系。显然，栅格表只能说明正在发生的情况，而无法提供应该采取的措施，这是它的局限性。

### 3. 战略目标集转换法

该方法由 W. King 于 1978 年提出，他把整个战略目标看成是一个“信息集合”，由目的、目标、战略和其他战略变量（如管理的复杂性、重要的环境约束）等组成，其基本出发点是将该集合转换为信息系统的目标与战略。

战略目标集转换法（Strategy Set Transformation, SST）是通过下面诸步骤形成信息系统战略规划。

（1）说明企业战略集合。具体为：说明企业相关人员的结构，如供应商、顾客、经理、雇员、股东、竞争者等；识别每类人员的目标；指出每类人员的任务及战略。

（2）请管理人员和高级领导人对形成的目标和战略进行审阅、修改，最后形成包含企业目标、战略和战略属性的企业战略集合。

（3）将企业的战略集合转化成信息系统战略规划。具体为：针对企业战略集合中的每个战略及相关目标与属性，找出一个或多个信息系统的目标；从企业的战略和信息系统的目标中找出信息系统的约束条件；根据企业的战略属性、信息系统的目标和信息系统的约束条件，找出信息系统的设计战略。

例如，某企业相关人员结构为股东、债权人、顾客等，企业的目标之一是改善现金流通。这是来自股东和债权人的目的。为达到这个目的，战略之一是改进信贷业务，由此引导出信息系统的目标是提高开单速度，这个目标的约束主要是计算机和决策模型，其战略可能是新的设计方法。这样，企业战略就转化成为信息系统的战略规划。

### 4. 企业系统规划法

企业系统规划法（Business System Planning, BSP）是一种对企业管理信息系统进行规划和设计的结构化方法，它由美国的 IBM 公司在 20 世纪 60 年代末创造并逐步

发展起来。BSP 法主要基于用信息支持企业运行的思想,是把企业目标转化为信息系统战略的全过程,BSP 法所支持的目标是企业各层次的目标,实现这种支持需要许多子系统。

BSP 是从企业目标开始,然后规定其处理方法,自上而下地推导出信息需求的。事务处理是数据收集和分析的基础。通过与经理面谈,弄清处理过程,并询问企业成功的关键因素,明确决策方法和问题,找出逻辑上相关的数据及事务处理的关系。这些信息可以用来定义未来的信息结构。根据当前系统和未来系统的信息结构,就可以建立应用的优先级别,并开始数据库的设计。可以将 BSP 看成是一个转化过程,即把企业的战略转化成信息系统的战略。该方法曾应用于我国经济信息系统的规划研究当中,并产生过很大的影响。

#### (1) BSP 的主要目标

BSP 的主要目标是提供一个管理信息系统规划,用以支持企业短期的和长期的信息需要,而且作为整个企业规划中不可缺少的部分。其具体目标可归纳如下:

- ① 为管理者提供一种形式化的、客观的方法,明确建立管理信息系统的优先顺序;
- ② 为具有较长生命周期的系统建设、保护系统的投资做准备;
- ③ 为了以最高效率和有效地支持企业目标,BSP 提供数据处理资源的管理;
- ④ 通过提供响应用户需求的系统,改善信息系统管理部门和用户之间的关系;
- ⑤ 将数据作为一种企业资源加以确定。

#### (2) BSP 的基本原则

① 一个信息系统必须支持企业的战略目标。基于这种思想,可以将 BSP 看成是一个转化过程,即将企业的战略转化成信息系统的战略。

② 一个信息系统的战略应当表达出企业的各个管理层次的需求。一般认为,在企业内同时存在着三个不同的管理层,即战略管理层、策略管理层和操作管理层。对不同层次的管理活动有着不同的信息需求,因此有必要建立一个合理的框架,并据此来定义信息系统。

③ 一个信息系统应该向整个企业提供一致的信息。由于各种单项数据处理系统的分散开发,形成信息的不一致性,包括信息形式上的不一致、定义上的不一致和时间上的不一致,所以为了保证信息的一致性,有必要制定关于信息一致性的定义、技术实现及安全性的策略与规程。

④ 一个信息系统应该经得起组织机构和管理体制的变化。信息系统应具有可变更性或对环境变更的适应性,即应当有能力在企业的组织机构和管理体制的变化中发展自己而不受到大的冲击。为了实现上述目的,BSP 采用定义企业过程的概念与技术,这种技术使信息系统独立于组织机构中的各种因素,即与具体的组织体系和具体的管

理职责无关。

⑤ 一个信息系统应是先“自上而下”识别，后“自下而上”设计。BSP对大型信息系统所采用的基本方法是“自上而下”地识别系统目标、识别企业过程、识别数据，和“自下而上”地分步设计系统，这样既可以解决大规模的企业信息系统难以一次设计完成的困难，也可以避免自下而上分散设计时可能出现的数据不一致问题、重新系统化问题和相互无关的系统设计问题。

### （3）BSP 的步骤

实施 BSP 是一项系统工程，其步骤大致分成如下几个阶段。

① 研究项目的确立：BSP 研究必须反映最高领导层对企业信息系统发展的想法，研究所提出的建议也必须得到最高领导层的批准，而一旦批准，企业在数年内就要按照规划所提供的方向去做。因此，BSP 研究一开始，就要得到最高领导参与研究的承诺，并对研究目标和应交付的成果取得一致意见。

② 研究准备工作阶段：研究项目确立之后，则应成立由最高领导牵头的委员会，下设一个规划研究组。其主要工作应当是研究计划的制定，内容包括研究计划、采访日程表、复查时间表、研究报告大纲及必要的经费确定。所有这些只有得到委员会的认可，准备工作才算落实。准备工作十分重要，大量事实证明，在未做好准备工作的情况下仓促上阵，结果是欲速则不达，危害整个工程。

③ 研究开始阶段：BSP 研究的首项活动是企业情况介绍。第一，重申研究的目标、期望的成果和研究的远景等；第二，介绍有关资料并讨论有关企业的决策过程、组织职能、开发策略等问题；第三，介绍数据处理部门的历史、现状、目前的主要活动和存在的主要问题。通过介绍，达到对企业和对信息支持的要求有较全面了解的目的。

④ 定义企业过程：定义企业过程是 BSP 的核心。企业过程是指在企业资源管理中所需要的、逻辑上相关的一组决策和活动，这些活动将作为安排与管理人员面谈、确定信息总体结构、分析现行系统、识别数据类及随后许多研究项目的基础。

⑤ 定义数据：定义企业过程后，则要识别和定义由这些过程产生、控制和使用的数据。数据类是指支持企业过程所必需的逻辑上相关的数据，并将数据按逻辑相关性归成类。

⑥ 分析现行系统支持：对目前存在的企业过程、数据处理和数据文件进行分析，发现缺欠和冗余部分，进而对将来的行动提出建议。

⑦ 确定管理部门对系统的要求：BSP 自顶向下的研究方法，决定了在整个规划过程中必须考虑管理人员对系统的要求，特别是对长远发展的看法。要通过与高层管理者的对话来明确目标、问题、信息需求和它们的价值，使 BSP 研究成员和管理部门间

建立新的、更密切的关系。

⑧ 提出判断和结论：通过采访可获得大量资料，这些资料随着研究组对企业了解的深入而进一步得到扩充和完善，最后通过与管理部門的会谈，对这些材料做出进一步的确认和解释。然后开始对问题进行分析，使用问题/过程矩阵等方法将数据和企业过程关联起来。通过关联分析，不仅为安排项目的优先顺序提供帮助，也将有助于解决信息系统的改进问题。

⑨ 定义信息总体结构：定义信息总体结构是由对现行情况研究转向对未来计划综合的主要步骤。信息总体结构刻画出未来的信息系统的框架和相应的数据类型。因此，该步骤的核心是子系统划分，具体实现时可以使用功能/数据类（U/C）矩阵。

⑩ 确定总体结构中的优先顺序：一个规模较大的信息系统的开发和实施，需要确定系统开发的优先顺序，即对信息总体结构中的子系统或其子项目根据所制定的准则进行重要性评价，从而排定开发顺序。

⑪ 评价信息资源管理：为了实现更完善的信息管理体系，使信息系统能有效和高效率地开发、实施和运行，需要对与信息系统相关的信息资源管理加以评价和优化，并使其能不断随企业战略的变化而改变。

⑫ 制定建议书和开发计划：建议书用来帮助管理部门对所建议的项目作出决策。项目是由信息总体结构优先顺序和信息管理部门的建议来决定的。开发计划则要确定具体的资源、日程，估计工作规模等。

⑬ 提交研究成果报告：最后要向最高管理部门提交 BSP 研究成果报告，报告中的各部分虽然是在不同阶段逐步完成的，但最后要将整个报告完整化和规范化。

#### （4）定义企业过程的方法

整个企业的管理活动是由一系列企业过程所组成的。定义企业过程可以帮助理解企业如何完成其目标，可以有效地支持所开发的信息系统结构独立于组织机构，为从操作控制过程中分离出战略规划奠定基础，为定义所需的信息结构提供依据。

定义企业过程，首先要识别企业过程的三类主要来源：计划/控制，产品/服务，支持资源。任何企业的活动均与这三方面有关并可由这三方面导出。定义企业过程的步骤如图 4.6 所示。

从第一类企业过程源——计划/控制，可以定义企业战略规划和管理控制方面的过程。战略规划方面的过程有经济预测、组织计划、政策开发、生产线模型等。管理控制方面的过程有市场/产品预测、资金计划、运营计划、人员计划等。

第二类企业过程源是产品/服务。由于任一产品都有其生命周期，即要求、获得、服务、退出，故在每一阶段都需要一些过程对其进行管理。如在要求阶段，所需过程为市场计划、预测、定价、材料需求计划等；在获得阶段，所需过程为产品说明、材

料购买、生产运行与调度等；在服务阶段，所需过程为库存控制、质量检测等；在退出阶段则为销售、运输等。因此，可以沿着生命周期去弄清这些过程。

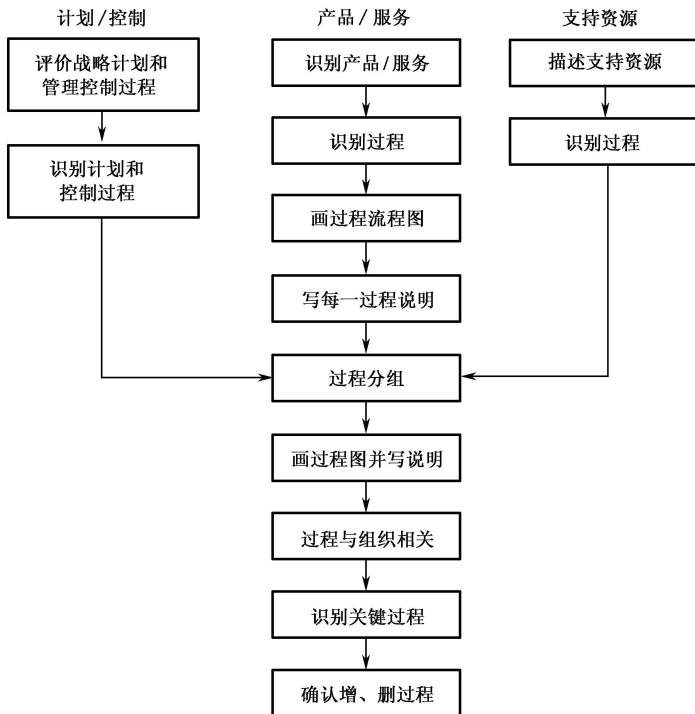


图 4.6 定义企业过程的步骤

第三类企业过程源是支持资源，其识别方法与产品/服务类似。一般，企业支持资源包括人、财、物，这些资源也都有其生命周期。对“人”而言，生命周期中的要求、获得、服务、退出的对应过程为人事计划、招聘、补充、退休；对“财”而言，对应过程为财务计划、资金获得、财务管理、会计支付；对“物”而言，对应过程为生产材料和设备需求、购买材料和设备、库存控制与机器维修、定货控制与设备报废等。

完成上述过程识别后，要进行过程分组，应合并同类过程并减少过程在层次上的不一致。然后画出过程组合表，即将每一过程组合和它的过程都列在一张表上。之后，再用建立过程/组织矩阵的方法，把企业组织机构与企业过程联系起来，从而说明每一过程与机构的联系。下一步开始识别关键过程，即识别企业成功的关键过程。识别关键过程是为了决定要对企业的哪些部门作更详细的研究。

定义企业过程是 BSP 成功的关键步骤。本步骤做完后应产生如下文档。

- ① 分别列出计划/控制、产品/服务、支持资源所导出的过程。第一类按战略规划和管理控制列出，第二、第三类按生命周期列出。
- ② 分别对每一过程作出简单说明。例如，对生产计划的说明为“为生产满足需求预测的产品，而对材料、人员、设备所进行的计划活动”。
- ③ 作出产品/服务过程的流程图。
- ④ 列出关键过程名。
- (5) 定义数据类的方法

数据类是指支持企业过程所必要的逻辑上相关的数据。定义数据类的第一步是识别数据类，其目的主要是：了解当前支持企业过程的数据的准确性和提供的及时性；识别在建立信息总体结构中要使用的数据类；发现企业过程间的数据共享；发现各个过程所产生、使用和缺少的数据等。

识别数据类的常用方法是企业实体法，即以企业实体为线索，通过其生命周期各阶段相关的数据的类型去识别数据类。图 4.7 表示了联系于各实体的生命周期阶段的各种数据类型。

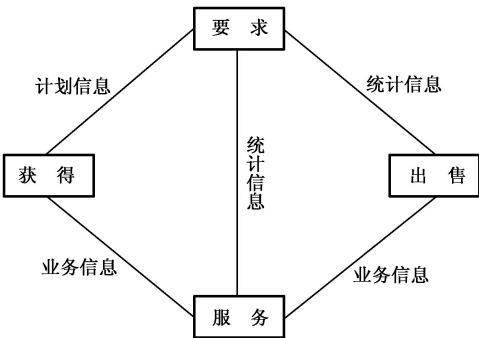


图 4.7 识别数据类

为进一步识别与企业实体有关的数据类，可以通过数据类型/企业实体矩阵，其中行表示主要的数据类型，列表示企业实体，每一单元的数据则表示对应每一种企业实体相对于每一个数据类型的数据类。表 4.1 列出了识别数据类的数据类型/企业实体矩阵示例。



表 4.1 数据类型/企业实体矩阵示例

企业实体 数据类型	产品	顾客	设备	材料	资金	人员
计划/模型	生产计划 质量计划	销售计划 市场计划	能力计划 设备计划	材料需求 材料采购	财务计划	人员计划
统计/汇总	质量汇总 入库汇总	销售历史 合同汇总	设备利用率	材料月耗 库存材料	财务统计	生产率 职工人数
文档	产品质量标准 成品质检报告	顾客档案 订货数据	设备使用 设备维修	原材料成 本材料单	财务会计总账	职工档案
业务	订货合同 提货单	发送数据	固定资产 设备购置	采购订货 入库出库	应收应付 采购借款	人事调动 劳动定额

数据类的最后确定，一般依赖于企业过程法。企业过程法是用来确定各个过程使用或生产了哪些数据类的。其具体工作是按产品/服务生命周期的顺序，构造一系列的输入—过程—输出数据类图，如图 4.8 所示。其中，每个输入和输出都是待定的数据类，要识别的是输入或输出数据类的实体。

在做完上述分析后，要写出每一个数据类的定义，并说明它包含什么数据，供讨论和定义数据结构使用。

BSP 将数据类和过程两者作为定义企业信息系统总体结构的基础，因此还要利用过程/数据类矩阵来表达两者间的关系。

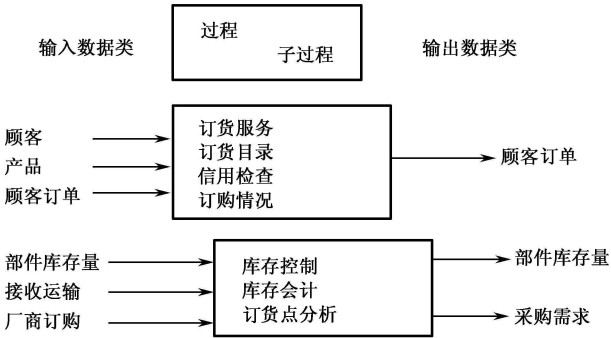


图 4.8 输入—过程—输出数据类图

（6）分析现行系统对企业的支持

本阶段的任务是要了解当前的数据处理工作对企业的支持，以便为未来信息系统的建设提出建议。



### (7) 定义信息总体结构

为识别要开发的信息系统及其子系统, 要用表达数据对系统所支持的过程之间的关系图来定义出信息总体结构。这种结构图应勾画出每一系统的范围, 产生、控制和使用的数据, 系统与系统的关系, 对给定过程的支持, 子系统间数据的共享等, 应成为企业长期数据资源规划的图形表示。目前最常用的方法是使用 U/C 矩阵。

U/C 矩阵将数据类作为列、功能(或过程)作为行, 用功能与数据类交叉点上的符号 C(Create) 表示这类数据由相应的功能产生, 用交叉点上的符号 U(Use) 表示这类功能使用相应的数据类, 空着不填的表示功能与数据无关。

在概括地介绍了 BSP 的基本概念和基本内容之后, 需要说明三点: BSP 适合较大信息系统的规划; 该方法本身是建立信息系统蓝图, 而不是详细设计; 目前存在许多 BSP 的变形方法, 也已取得一定的应用效果。

## 5. 关键成功因素法

在每一个企业中都存在着对该企业成功起关键性作用的因素, 称为关键成功因素。关键成功因素总是与能确保企业具有竞争能力的方面相关。在不同类型的业务活动中, 关键成功因素会有很大的不同, 即使在同一类型的业务活动中, 在不同的时间内, 其关键成功因素也会不同。在多数企业中, 通常有 3~6 个决定企业成功与否的因素。

关键成功因素与企业战略规划密切相关。企业战略规划要描绘企业的期望目标, 关键成功因素则提供了达到目标的关键和需要的测量标准。一个企业要获得成功, 就需要对关键成功因素进行认真的和不断的度量, 并时刻注意对这些因素的调整。关键成功因素法(Critical Success Factor, CSF)就是帮助识别关键成功因素的方法, 它在确定企业关键成功因素和信息系统关键成功因素方面都收到了较好的效果。其使用通常包含以下几个步骤。

(1) 了解企业(或信息系统)的战略目标。

(2) 识别所有成功因素。可采用树枝图, 画出影响战略目标的各种因素及影响这些因素的子因素。

(3) 确定关键成功因素。对所有成功因素进行评价, 根据企业现状与目标确定出关键成功因素, 这时可采用德尔斐法、模糊综合评判法等。如前所述, 不同企业的关键成功因素是不相同的。

(4) 识别性能指标与测量标准。给出每个关键成功因素的性能指标与测试标准。

例如, 某企业有一个目标是提高产品竞争力, 可以用树枝图画画出影响它的各种因素及影响这些因素的子因素, 如图 4.9 所示。

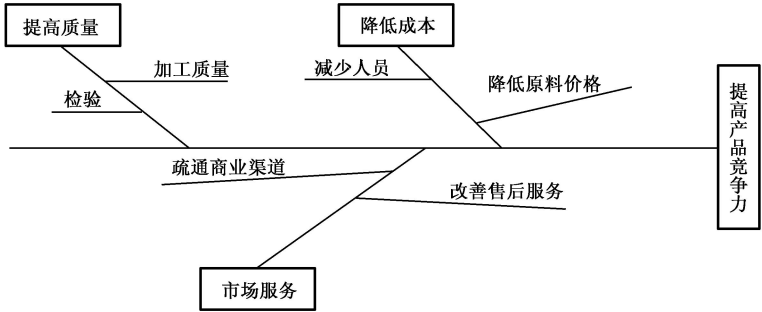


图 4.9 树枝图

关键成功因素法在高层的应用效果较好，因为高层领导人员日常总在考虑什么是关键成功因素；对中层领导来说一般不大适合，因为中层领导所面临的决策大多数是结构化的，其自由度较小。

6. 目的手段分析法

目的手段分析法是一种适合于信息系统需求分析的方法。这种方法首先着眼于组织过程的目的或输出。输出可以是产品、服务及信息，而达到这些目的的手段则包括输入和过程。一个过程，其结果或输出总是另一个过程的输入。例如，存储过程向生产过程提供零件，会计过程向其他过程提供预算信息等。因此，这种分析可以反复地迭代进行。

7. 投资回收法

投资回收法是一种成本/效益分析法。这是一种经典的决策方法。在计划每一个信息系统的应用项目前，要将其经济成本和经济效益定量表示出来，利用这两个量就可以计算出投资回收率。如果有多个项目开发方案，则可选取投资回收率最高的项目。

具体使用该方法时有一定限制，主要有下面三点：一些效益是非直接的、隐性的或综合的，因而很难量化；仅仅依据投资回收法确定的优先开发项目不能提供在风险等方面的情况，因而这种方法缺乏综合性；投资回收率的计算方法是渐进式的，不能引起人们对当前应用项目的重新思考。

8. 零点预算法

零点预算法是一种可用于信息系统资源分配的方法，其步骤大体分为如下三步：

设想所有的信息系统工作都从零开始,即还未开发,更没有维护问题;列出所有潜在的信息系统应用项目,并按服务层次进行分类,对每一层次列出其期望效益和所需要支持的资源,形成一张总清单,递交指导委员会或资源分配机构来决定优先次序;每一项目都是从应用的零点基础开始的,而且当项目及其功能需增加时,将被给予不同的优先级别。通过讨论,确定出应用项目的优先次序并计算出资源需要。

## 4.4 信息工程与战略数据规划

### 4.4.1 信息工程的基本原理

信息工程 (Information Engineering) 是在多年信息系统研究的基础上总结其成功和失败的经验,进行了一系列实际考察和理论分析,并综合了多种方法,于 20 世纪 80 年代初产生的一套理论与方法。根据 James Martin 的意见,“信息工程作为一个学科要比软件工程更为广泛,它包括了为建立基于当代数据库系统的计算机化企业所必需的所有相关的学科”。

信息工程的基本原理如下所述。

(1) 数据位于现代企业数据处理的中心。这就是说,企业所运行的各种应用软件都应围绕着数据来展开,数据管理是系统建设的核心。

(2) 数据是相对稳定的,处理是多变的。也就是说,一个企业所使用的数据的类型和结构是相对稳定的,除了偶尔少量加入一些新的实体外,变化的只是这些实体的属性值。而数据处理过程的变化则是频繁和快速的,数据管理人员需要最大的灵活性,以保证数据处理过程能适应管理者快速多变的信息需求。如果基本数据的结构已经建立,就可以使用高级数据库语言和应用生成器,很快地建立企业的数据处理过程。

(3) 全面地进行数据规划是系统建设的根本所在。进行数据规划是信息系统开发策略的需要,它关系到系统开发的整体性和一致性,也是合理制定整个系统实施计划的前提。没有全面的数据规划,各子系统独立实施的结果是难以组成协调的大系统的。当这种协调必须进行的时候,则需要各子系统加以转换,而完成这种转换的代价是昂贵的。

(4) 用户必须直接参与信息系统的建设工作。James Martin 指出,企业信息系统的开发工作能否成功,主要取决于管理者对本企业活动的看法、对信息系统的需求程度及他们能否使研究人员理解企业的业务活动。而且,信息系统运行后的信息输入也都直接或间接地来源于这些管理者,信息系统的输出信息也都直接或间接地为这些管理者所用。如果企业信息系统的总体数据规划仅由数据处理部门提出,那么,一是数据处理部门没有足够的权力来规定统一的数据定义方式,二是数据处理专家也无法对

企业的业务活动有充分的理解，因而往往难以满足管理上的要求。

(5) 自顶向下规划与局部设计相结合。为了使规划工作顺利展开，应该建立合适的工作组，工作组必须由经验较丰富的成员组成，可从本企业或外面的咨询机构中选取。外来顾问必须是能提供一套成熟的、得到验证的科学方法的专家。对于大型企业，要完成一个自顶向下的规划设计，核心小组应包括数据处理管理人员、系统分析人员、资源管理人员、财务总监、企业的业务经理、客户服务经理等。核心小组成员应由外来顾问进行培训和指导。

图 4.10 说明了在信息资源规划工作中，信息资源规划者自顶向下的规划和数据管理员自底向上进行详细设计工作之间的相互关系。自顶向下的规划者应着眼于全企业，决定企业需要的数据库和其他数据资源。数据管理员则对收集的数据进行分析并综合成所要建立的每个数据库。

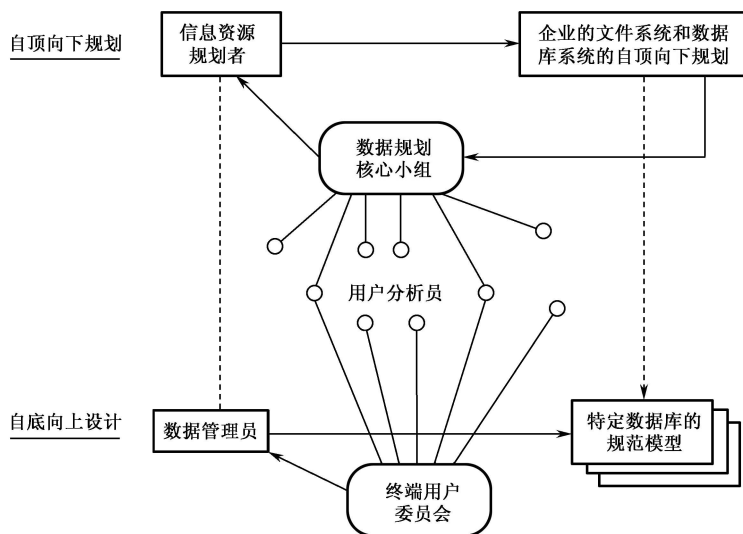


图 4.10 自顶向下规划与自底向上设计

信息资源规划者与数据管理员都需要终端用户的帮助，但他们所需的提供内容不同。信息资源规划者要求各个职能部门用户所提供的信息和数据不必太详细；数据管理员则要求终端用户在一段时间内对每一个主题数据库进行详细的精确的审查，力图使这些主题数据库保持尽可能的稳定。

主题数据库是战略数据规划的重要内容之一。主题数据库与企业的经营主题有关，而不是与一般的应用有关。例如，在一个企业中，可建立数据库的典型主题有产品、

客户、零件、顾客、订货、账目、人事、档案、工程规范等。

数据的全局规划是十分重要的，但真正要实现一个完整的、统一的整体数据库却是不现实的。因此，应采取自顶向下的方法进行数据规划，确保信息的一致性，然后在其框架内进行自下而上的详细设计，并充分利用开发工具的支持。

#### 4.4.2 信息工程方法论

信息工程方法论包括一整套建设信息化企业的方法，可以概括为三部分。

##### 1. 关于信息系统设计实现的方法

信息工程方法将信息系统的开发周期分为八个阶段，如图 4.11 所示。

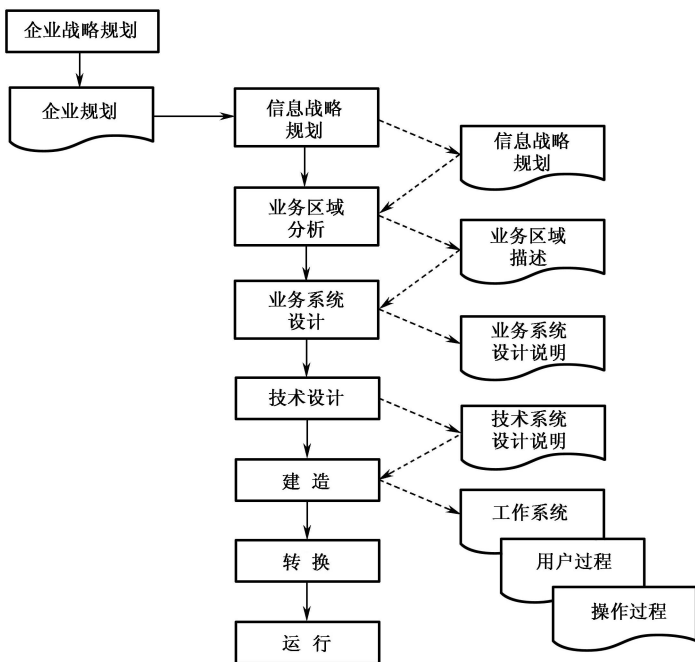


图 4.11 信息系统开发阶段及成果

(1) 企业战略规划：涉及企业全面的业务规划，包括企业的目标和任务。

(2) 信息战略规划：从企业目标出发为企业开发一个基本的信息体系结构。

(3) 业务区域分析：对规划中所识别区域的组成进行分析，确定出它们之间的所有联系和相互作用，开发出详细的企业模型。

- (4) 业务系统设计：确定需要计算机支持的业务过程和活动。
- (5) 技术设计：提供全面的输入/输出格式、数据库结构等。
- (6) 建造：实现程序设计和数据库建立，完成整个系统的测试和运行准备。
- (7) 转换：指现行信息系统向新的信息系统的过渡。
- (8) 运行：指系统的整个运行生命周期，包括系统的性能监测、维护。

## 2. 关于企业信息系统战略规划的方法

关于企业信息系统战略规划的方法主要是战略数据规划，也称为总体数据规划，它是信息工程规划工作的基础与核心。4.4.3节将介绍该战略数据规划方法的目标与步骤。

## 3. 关于自动化开发工具

信息工程方法论强调了开发工具的重要性。常见的开发工具有两种主要模型：面向处理的模型（Process-Oriented Model）和面向对话的模型（Dialogue-Oriented Model）。在面向处理的模型中，系统被看作是一组分层的处理过程，通常使用结构化的方法进行变换。这种模型具有整体性和结构统一性的优点，但是它对那些未经程序设计训练的开发者来说太抽象了，难以掌握。在面向对话的模型中，系统被看作是一系列连接计算机和操作员的对活，通常建立各种对话交互过程。这种模型对系统开发者更简单，但是如果应用项目本身很难建立在屏幕交互上，则它们就可能缺乏一般性。除了这两种主要模型外，还有面向过程的模型、面向文件的模型和面向报表的模型等多种模型环境。

## 4.4.3 战略数据规划的目标与步骤

### 1. 四类数据环境与战略数据规划的目标

James Martin 将计算机的数据环境分为四种类型，并指出，一个高效的企业应该基本上具有三类或四类数据环境作为基础。

#### (1) 第一类环境：数据文件

① 特征：没有使用数据库管理系统。当建立一个应用项目时，由系统分析员或程序员分散地根据应用需要设计各种数据文件。大多数应用项目都使用这类独立的文件。

② 特点：建立过程相对容易，但当数据文件的数目剧增时，将导致维护成本高。

#### (2) 第二类环境：应用数据库

① 特征：虽使用了数据库管理系统，但没有达到第三类数据环境的那种共享程度，

分散的数据库为分散的应用而设计。

② 特点：实现起来比第三类数据环境简单，但像数据文件环境一样，随着应用的扩充，其维护费用是很高的。

### （3）第三类环境：主题数据库

① 特征：经过了严格的数据分析，所建立的数据库与具体的应用是独立的，数据的存储结构与使用它的处理过程是独立的。

② 特点：这种数据环境的建立具有较低的维护成本，但需要详尽的数据分析和模式化。如果管理不善，则会退化成第二类，甚至第一类数据环境。

### （4）第四类环境：信息检索系统

① 特征：这是一类为自动信息检索、决策支持系统和办公自动化所设计的，而不是为专用计算和事务管理所设计的系统，它能保证信息检索和快速查询的需要，如新的数据项可随时动态地加入到数据库中，其软件是围绕着倒排表和其他的数据检索技术设计的，并有良好的终端用户查询语言和报告生成软件工具，由此可灵活地创建用户自己的逻辑数据文件。

② 特点：比传统的数据库系统有更大的灵活性和动态可变性，通常与第三类数据环境并存。

对于目前只有第一、二类数据环境的组织，通过战略数据规划，可尽快将现有数据环境转变到第三、四类，以保证高效率、高质量地利用数据资源。对于目前还没有数据环境的组织，战略数据规划是整个计算机信息系统应用发展规划的基础与核心，是制定设备购买规划、人力培训规划和应用项目开发规划的基础。通过战略数据规划，可选择最适宜的数据环境。概括地说，企业制定战略数据规划的主要目标就是分析、组织、建立企业稳定的数据结构，规划各种主题数据库的实施步骤和分布结构，为企业信息管理计算机化打下坚实的基础。

## 2. 战略数据规划的基本步骤

战略数据规划可分为六步，亦称为六个阶段。

### （1）第一个阶段：建立企业模型

企业模型表示了该企业在经营管理中具有的功能。不同的企业模型对企业活动表示的详细程度各异。企业模型根据数据需求实现面向数据的变换，这个变换可分解成需要实现的多个数据库。企业模型的建立大致可分为三个部分：开发一个表示企业各职能范围的模型；扩展范围模型，使其能表示企业各处理过程；继续扩展范围模型，使其能表示企业各业务活动。

企业模型应具有以下特性。



① 完整性：模型应提供包括企业的各个职能范围、各种处理过程、各种业务活动的一个完整的图表。

② 适用性：模型应是人们合理地、有效地理解企业的途径，在分析中所确定的过程和活动应是自然和确切的。

③ 持久性：只要企业的目标保持不变，模型就应该保持正确和有效。只要企业执行的职能相同，模型就依然适用。

企业模型化的过程经常会揭示出企业组织机构中的一些冗余和异常的情况，因此可能导致整个企业的重组和调整。

#### （2）第二个阶段：确定研究边界

当一个企业的各个职能被表示出来后，就必须确定合适的研究范围或边界。在一个小型企业或密集型的一体化企业中，研究的范围包括整个企业；在一个联合企业中，应先在一个公司内进行规划，并利用所取得的结果指导其他公司的规划工作。

战略规划的研究范围与企业的管理方式有关。有些企业的部门相对独立和自治，另一些企业则情况相反，即使在相同的工业领域或相似的企业，其管理方式也可能不尽相同。在采取分散管理方式的企业中，自顶向下的全局规划可在几个不同的部门各自独立地进行，而在采取集团管理方式的企业中，自顶向下的全局规划需要同时涉及所有的部门。全局规划的边界划定要适当，不要太大。

#### （3）第三个阶段：建立业务活动过程

当企业模型建立后，就应着手建立每个职能范围所包含的业务过程。通过对每个职能范围有代表性活动过程的分析，可以确定各个业务活动过程。建立业务活动过程，主要依靠企业高层领导和各级管理人员，分析企业的现行业务和长远发展目标，按照企业内部各种业务的逻辑关系，将它们划分为若干职能区域，然后建立各职能区域中所包含的全部业务过程。

#### （4）第四个阶段：确定实体和活动

职能范围和业务活动过程确定后，则着手进行实体和活动的确定，细化有关实体的数据及与之相关的活动，形成一个更详细的企业模型。

确定实体和活动的方法是培训企业中各个不同职能范围内的用户，在系统分析员的指导下由用户来完成这项工作。

一般，在信息系统规划中大量采用直观的图形或图表表示方法。全局规划图要细化到能正确指导数据库和系统设计。每个职能范围的工作模式应交给各职能部门的业务人员进行审查。

#### （5）第五个阶段：建立主题数据模型

在上述业务分析过程中，已弄清所有业务活动所涉及的数据实体及其属性，本阶



段重点分析这些实体之间的联系，即按照管理人员的经验和一些形式化方法，对实体进行聚集分析，以此作为划分主题数据库的依据，建立起主题数据库模型，即针对企业的职能区域和业务过程提供必要的数据共享的总体数据模型。

#### (6) 第六个阶段：分析数据分布

结合数据存储地点，进一步调整、确定主题数据库的内容和结构，制定数据库开发策略。数据分布分析要充分考虑业务数据的发生和处理地点，权衡集中式数据存储和分布式数据存储的利弊，考虑数据的安全性、保密性、运行效率、用户特殊要求等问题。

## 4.5 可行性研究

可行性研究又名可行性分析、可行性论证，是信息系统开发的最初的重要环节，它关系到系统开发使用的成败。任何大型项目在正式投入建设之前都必须进行可行性研究，不允许盲目上马，否则必然会造成巨大的浪费，也可能还未建成就不得不下马了。

对于信息系统开发而言，可行性研究的目的是解决“是否可能”和“有无必要”的问题，其基本含义是：根据组织当前的实际情况和环境条件，从各方面来判断这个信息系统的建立是否必要，以及是否具备开发所需的资源条件。可行性研究阶段所编制的可行性研究报告，一般作为“立项”和申请经费、设备、人员等资源的主要依据，也是上级管理人员进行科学决策的主要依据。

然而，由于社会经济系统的复杂性，到目前为止，还没有一套完整的、通用的可行性研究的方法。因此，只要能够完成可行性研究这项工作任务，能够回答“是否可行”和“何者最优”这两个问题的一切科学方法和传统方法（经验）都是可用的。

可行性研究工作从接受任务开始，一直到通过可行性研究报告为止，可以大致分为明确要求、环境调查、提出方案、可行性分析等步骤。

### 4.5.1 明确要求

提出开发信息系统要求的可能是企业部门的领导，如管理职能处（科）室的负责人，也可能是企业的高层管理人员，也可能是上级机关。他们提出开发要求的原因可能是：

(1) 信息处理的能力不能适应管理工作的需要，不能及时向管理决策提供必要的信息；

- (2) 数据重复收集和存储,增加了管理工作量和出错的可能性;
- (3) 难以满足随机的和突发性的查询统计要求;
- (4) 信息利用率不高,分析综合工作不足;
- (5) 乏味、重复、烦琐的手工处理方式。

可行性研究要确定系统的目标、要求、任务、功能等。这些首先由用户提出,他们的原始要求很重要,特别值得重视。但是,往往由于他们对计算机能做什么并没有确切的概念,更没有定量的标准,因此一般说来原始要求是含糊的、不具体的和不确切的,有时主次不分,只罗列问题和要求。所以,一般说来,用户的最初陈述只是提供了形成系统目标的素材,而不是明确的系统目标和功能,不能作为判断信息系统开发是否可行的依据。这种情况的出现是必然的,我们不能要求企业的领导人和管理人员都通晓计算机信息处理的细节,把他们的要求明确化、量化、分清主次而形成科学的项目目标。这些必须由系统开发人员来完成。

在明确要求的过程中,系统开发人员要在调查研究的基础上,根据实际情况,认真分析用户所提出的一系列要求和解决问题之间的关系,这些要求之间可能存在某些因果关系,也可能存在着互相矛盾、需要加以权衡的关系,有的还需要确定主次顺序,只有把这些关系分析清楚,才能抓住实质,明确地表达出项目的主要目标和要求。最后,系统开发人员应该使用非技术性语言、以书面的形式描述系统的目标,新系统的功能范围,要求的时间进度,可能投入的人力、物力、财力资源,以及其他关键性问题。

#### 4.5.2 环境调查

环境调查的目的是对企业的环境给出一个概括性的描述,以便提出候选方案和进行可行性分析。调查的重点是企业组织与原信息系统的总的情况、企业的外部联系、企业的能力和发展规划,以及企业的各种资源条件和受到哪些外界条件的限制,因为这些因素是系统开发人员所不能左右的而必须在工作中服从的因素,它对可行性研究的影响最大。同时,还应在调查中,重点了解现行信息系统存在的问题和要求,作为新系统开发的出发点。

企业环境调查主要包括以下几个方面的内容:企业概况、组织结构与外部联系、主要业务流程、当前系统的现状、主要问题、设备能力、库存能力、财务成本等。

在环境调查时所采用的方法常常是阅读资料,以及与企业领导和有关部门的领导进行面谈或座谈,也可根据情况设计各种调查表辅助调查。环境调查所投入的人力不必太多,但是要求这些人应具有相当的工作经验。

### 4.5.3 提出方案

在环境调查的基础上,根据用户提出的要求,对建设新系统的要求作出分析和预测,同时考虑新系统所受到的各种制约因素,根据需求和可能,给出几种拟建系统的候选规模和方案。

#### (1) 分析用户要求,确定系统目标

目标指的是在一定的时期内的努力方向和要达到的具体指标。从期限上分,目标可分为长期目标、中期目标和短期目标;从经济上分,可分为生产目标、销售目标、利润目标、新产品开发目标、质量目标等;从管理层次上分,可分为企业整体目标、管理层次的目标及基层生产目标等。可行性研究目标,只有根据用户提出的目标和对信息处理的要求,通过调查和分析才能确定。在系统总的目标确定以后,再将其逐个分解成若干个具体目标,画出系统的“目标树”,分析目标与客观手段、原因及结果等关系。

需求的分析应注意两点:一是必要性,二是现实性。

可行性研究阶段确定的系统目标只可能是粗线条的,进一步的细化要到系统分析阶段。

#### (2) 确定新系统的职能结构

确定新系统的职能结构一般是指子系统的划分。新系统按其功能可分为若干子系统,子系统的划分方法,通常有水平(横向)划分垂直(纵向)划分两种方法。水平划分是根据管理的级别将系统划分为战略管理、战术管理和作业管理;垂直划分是根据管理的对象、管理的职能将系统划分为计划管理、生产调度、财务管理、物资管理、销售管理、技术管理、人事管理、质量管理、设备管理、库存管理等。

一般地讲,子系统划分的原则是:

- ① 基于信息系统的职能,各子系统相对独立地完成部分管理功能;
- ② 各子系统应按业务信息的逻辑方式划分;
- ③ 各子系统间边界清楚,子系统内业务及数据联系紧密;
- ④ 子系统的划分应尽量考虑不受管理体制变化的影响,应适应机构改革。

子系统划分的目的是减少信息重复和交叉,提高信息处理效率,促进管理科学化。

子系统划分以后,应明确各子系统的功能、任务、相互关系,以及各子系统与现行管理部门各组织机构的关系。

#### (3) 确定新系统的数据结构

确定新系统的数据结构是指合理划分新系统的数据库。数据库是信息系统的重要

组成部分，从某种意义上讲，信息系统是由不同种类的数据库组成的数据库应用系统。

在本阶段，通过调查应确定新系统的数据结构，包括新系统需要存储的各类数据库（特别是公用数据库）的名称、所包含的主要内容、信息量的估计等；还要明确各类数据库和各子系统之间的关系，确定各子系统专用数据库的存储量和各主要子系统之间的信息交流关系及信息量。

#### （4）确定新系统的物理结构

在确定了新系统的目标、功能及数据要求以后，可以明确新系统的基本物理结构方案，如分布式结构方案、集中式结构方案等，确定计算机的选型、网络选型、软件配置、硬件配置，比较基本物理结构的优缺点、选型方案的优缺点等。

#### （5）确定新系统的开发进度

明确系统的开发总进度，各阶段进度、各子系统和各数据库的开发进度、以及讨论影响开发进度的主要原因和保障措施等。

在估计开发进度和开发周期时，相对来说最好是偏“松”一些。特别是所开发的信息系统规模大、涉及面广和缺乏经验时，常常可能出现不少“不可预见”的情况和困难。预计的周期太短，将影响系统开发人员的信心和造成一些人为的困难。

#### （6）经费预算和投资方案

对候选方案要作出经费预算，并提出投资计划（一般多为逐年投资）。新系统开发所需费用主要为以下几个部分：计算机设备和网络设备费用、系统软件和应用软件费用、新建或改造计算机房费用、系统开发费用、系统运行费用、人员培训费用、维护费用及其他费用等。

对于企业开发信息系统来说，其投资来源主要有企业自筹资金、各类贷款等。

### 4.5.4 可行性分析

可行性分析是在提出候选方案后，对方案进行包括经济可行性、技术可行性、法律可行性和开发方案的选择性等方面的分析。

#### 1. 经济可行性分析

经济可行性分析是指根据用户提出的系统功能、性能及实现系统的各项约束条件，从经济的角度研究实现系统的可能性。

成本效益分析是经济可行性分析的重要方法，它用于评估信息系统的经济合理性。给出系统开发的成本论证，并将估算的成本与预期的利润进行对比。由于项目开发成本受项目的特性、规模的制约，事先很难直接估算信息系统的成本和利润，因此得到

完全精确的成本效益分析结果是十分困难的。

系统成本前面已简单介绍过。系统效益包括经济效益和社会效益两部分。经济效益指应用系统为用户增加的收入，它可以通过直接的或统计的方法估算。社会效益只能用定性的方法估算。

成本效益分析还应该研究附加效益和追加成本之间的关系。

## 2. 技术可行性分析

技术可行性分析是指根据用户提出的系统功能、性能及实现系统的各项约束条件，从技术的角度研究实现系统的可能性。

技术可行性分析往往是系统开发过程中难度最大的工作。技术可行性分析包括风险分析、资源分析和技术分析。风险分析的任务是，在给定的约束条件下，判断能否设计并实现系统所需功能和性能。资源分析的任务是，论证是否具备系统开发所需的各类人员（管理人员和专业技术人员）、软件、硬件和工作环境等。技术分析的任务是，当前的科学技术是否支持系统开发的全过程。

在技术可行性分析过程中，系统分析人员应采集系统性能、可靠性、可维护性和可生产性方面的信息，分析实现系统功能和性能所需要的各种设备、技术、方法和过程，分析项目开发在技术方面可能担负的风险，以及技术问题对开发成本的影响等。数学建模、原型建造和模拟是信息系统技术分析活动的有效工具。

## 3. 法律可行性分析

研究在系统开发过程中可能涉及的各种合同、侵权、责任及各种与法律相抵触的问题。

## 4. 开发方案的选择性研究

在分析、评价的基础上，对所提出的各种系统开发方案进行综合性评估，从中选出一种用于项目开发。由于每种方案对成本、时间、人员、技术、设备等都有不同的要求，因此不同方案开发出来的系统在系统功能和性能方面会有很大的差异。同时，在开发系统所用总成本一定的情况下，系统开发各阶段所用成本分配方案的不同也会对系统的功能和性能产生相当大的影响。另外，系统功能和性能也是由多种因素组成的，某些因素是彼此关联和制约的。以上原因充分说明，系统开发方案的选择性研究很大程度上是对系统开发活动中多种因素的权衡、折中。

利用折中手段选择系统开发方案时应充分论证、反复比较各种方案的成本效益。折中过程也是系统论证和选择、确定系统开发方案的过程。

## 4.6 基于体系结构的信息系统顶层设计

### 1. 顶层设计的概念

顶层设计是在大系统建设的早期，确保满足全系统综合集成需求的关键环节。顶层设计的主要任务是依据需求，自顶向下地进行系统建设的统筹考虑，从系统战略全局和总体的高度，综合分析当前和未来的能力要求，综合考虑现实和未来对信息系统建设发展的要求，兼顾技术、经济等因素对信息系统建设发展的约束，明确信息系统建设发展的目标任务，审视、分析并准确刻画业务、信息、技术和应用系统间的相互作用关系，建立由多个层面共同构成的系统总体架构，并依据总体架构制定系统的发展战略、建设规划计划和工程管理措施。同时，以发展战略和总体规划为基础，对各种具体系统或项目进行规划论证，制定系统设计和开发的总体方案。

### 2. 顶层设计的思路和方法

顶层设计的主要思路是：强调综合化、网络化、一体化，把信息系统建设的需求、数据、标准、技术、构成等分析清楚，按照目标—需求—要素—结构—功能的建设思路和要求，形成信息系统建设的整体框架，为信息系统的建设提供科学的路线图。

近几年来，信息系统顶层设计的问题得到了普遍的重视，也进行了不少的探索和尝试，在开展制定建设发展规划、计划、总体方案等工作的同时，注重引入一些新的方法、技术和机制，并取得了一定的效果，且在体系性、客观性、合理性和可发展性等方面有所提高。但总的看来，这些顶层设计实践大多由于缺乏科学规范的方法、步骤和工具支持而并未起到应有的作用，可操作性不强，对具体系统建设的指导和约束作用缺乏针对性和有效性。

目前，顶层设计尚未形成一个公认的统一的方法体系。以美国为首的信息化发达国家，总结出了一套当前国际通行的顶层设计方法——体系结构方法。体系结构方法是当今世界广泛应用、普遍验证的顶层设计方法。从国外对体系结构的定义可以看出，体系结构实际就是一种系统的蓝图。该蓝图主要描述两个方面的内容。一方面是描述出系统的各组成部分的结构及它们之间的各种关系，另一方面是描述出系统上述结构和关系随时间的变化和演进关系，以及指导其变化和演进的原则与指导思想。

J. A. Zachman 于 1986 年首先提出了 Zachman 框架，在 Zachman 框架的基础上，先后出现了 FEAF 框架（Federal Enterprise Architecture Framework，联邦企业体系结构框架）、TEAF 框架（Treasury Enterprise Architecture Framework，财政企业体



系结构框架)和C4ISR体系结构框架等,并且有一些商业软件支持这些框架的设计和实现。目前,这些框架在相关领域中得到了广泛的应用,大大提高了这些领域内系统顶层设计质量和系统间的互操作能力。

### 3. 国外基于体系结构的顶层设计的现状及启示

目前,美国已经步入了体系结构的发布(完善)、评估和运用阶段。基于体系结构的顶层设计在美国政府和军队、企业和商业界已经广泛应用,并取得了相当显著的经济、社会和军事效益。

随着对信息系统综合集成要求的不断提高,以美国为首的西方国家普遍认识到,以体系结构为基础实现各级各类相关信息系统综合集成是满足一体化建设要求的必由之路。因此,他们对体系结构的开发法规规范、基础资源与工具的建设非常重视,投入了大量的人力和财力。

第一次海湾战争以前,美军各级军事信息系统都是由各军兵种独立建设的,由于对系统体系结构设计的重视程度不够,不同体系结构的表示方法各不相同,引用的术语也不一致,所以难以实现不同信息系统之间的集成,影响了信息系统体系效能第一次海湾战争中的发挥。海湾战争之后,美军在加快军兵种信息系统综合集成的同时,加强了顶层设计和通用基础资源的建设。

基于体系结构的顶层设计理论和方法有效地推进了美军一体化信息系统的规划、设计和建设工作,对美军的军事转型发挥了关键性的作用,主要表现在:一是指导现有信息系统的改造和升级,促进一体化信息系统的综合集成;二是指导大型一体化信息系统的研制、设计和建设。

从美军军事信息系统体系结构工程的建设过程,我们可以得出以下启示。

一是高度强调顶层设计的重要性,也充分验证了体系结构在大型综合信息系统顶层设计中的有效性和实用性。只要这样,才能确保不同的管理部门、不同的使用部门、不同的研制部门所建设的系统最终具有高度的互操作能力,并在使用中充分获取信息优势。

二是强调信息化建设需求的重要性,依据信息化建设目标及能力和功能上的需求牵引顶层设计,为确保未来研制的信息系统满足业务使用要求提供了保证。

三是以强制的形式规定必须为大型信息系统设计体系结构,而且设计时必须遵循统一的描述、过程和管理规范。严格按照相关法规规范设计的体系结构有力地指导了美军军事信息系统的建设,另一方面,美军通过体系结构工程实践不断地验证、修改和完善有关体系结构法规规范。因此,我们在信息化建设中要加强对信息系统体系结构作用的认识,采取强制措施开发各级各类信息系统体系结构。

四是顶层设计中法规规范与工程方法、支撑工具之间的关系是相辅相成的，在顶层设计实施过程中，需要重视对法规、规范和工具的建设。

## 习 题

1. 阐述信息系统战略规划对信息系统开发建设的作用和意义。
2. 阅读关于信息系统体系结构的文献，谈谈自己的认识。
3. 信息工程的思想本质是什么？
4. 可行性研究由哪几个过程组成？说明可行性分析的几个分析方面是什么？可行性研究的作用怎样？
5. 企业系统规划（BSP）方法的 13 个步骤是怎样的？其中最重要的步骤是什么？



## 第 5 章 结构化系统分析与设计

系统分析 (System Analysis)，也称需求分析，是信息系统开发工作中重要的、必不可少的环节，特别是针对中、大规模的信息系统开发，系统分析工作做得好坏，直接影响整个系统的成败。这一阶段是整个信息系统建设的最关键阶段之一，也是信息系统建设与一般基建项目的重要区别所在。

在较大规模的信息系统开发过程中，按照信息系统生存周期的观点，首先进行可行性研究，待可行性研究报告经评审批准后，转入系统分析阶段。系统分析是在可行性研究报告所限定的建设范围内，针对现行系统全面的调查分析，从而确定整个系统将“做什么”。

从系统的观点出发，系统分析就是对要解决的问题及其环境进行分析综合，找出各种可行方案，以提供给决策者进行选择的一种方法。系统分析建立在对问题及其环境深入调查研究的基础上，以系统的目标为依据，通过分析系统的结构、元素、输入及输出关系，建立能准确描述系统的逻辑模型。

系统分析之后将进入系统设计阶段。从方法论上考虑，进行系统分析和设计的方法主要有结构化系统分析与设计方法、面向对象系统分析与设计方法等。本章讨论结构化方法，下一章讨论面向对象方法。

### 5.1 结构化方法的基本思想

结构化方法 (Structured Method) 是最早的、最传统的软件开发方法。20 世纪 60 年代初，就提出了用于编写程序的结构化程序设计方法，而后发展到了用于设计的结构化设计方法 (Structured Design, SD)、用于分析的结构化分析方法 (Structured Analysis, SA) 及结构化分析与设计技术 (Structured Analysis and Design Technique, SADT) 等。

结构化方法的基本思想可以概括为：自顶向下、逐步求精；采用模块化技术、分而治之的方法，将系统按功能分解成若干模块；模块内部由顺序、分支和循环等基本控制结构组成；应用子程序实现模块化。

结构化方法强调功能抽象和模块化，将问题求解看作是一个处理过程。结构化方法由于采用了模块分解和功能抽象，自顶向下、分而治之的手段，从而可以有效地将

一个较复杂的系统分成若干易于控制和处理的子系统，子系统又可以分解成更小的子任务，最终的子任务是一些功能明确、结构独立的模块。这些模块功能相对独立，接口简明，界面清晰，使用和维护起来非常方便。所以，结构化方法是一种非常有用的开发方法，也是其他方法学的基础。

结构化分析与设计是结构化方法的最关键的两个时期，其主要任务是分析系统的功能、性能、目标、规模等需求，定义系统的逻辑模型；设计系统的模块结构、输入/输出、数据库、数据文件等；给出模块说明和主要算法；为以后的编码实现作算法上和结构上的准备。

在结构化方法的生存周期内，系统分析和设计的基本阶段可以划分为系统分析和系统设计两个阶段，如图 5.1 所示。

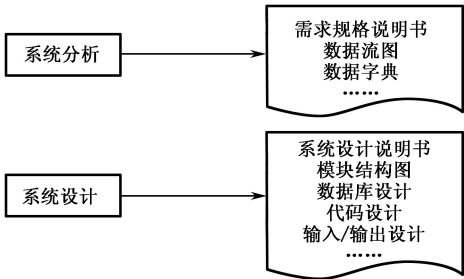


图 5.1 结构化分析与设计过程

结构化方法的缺陷是将过程和数据分离为相互独立的实体，从而造成了实现时的相对困难，对于不同的数据格式作同样的处理或对于相同的数据格式作不同的处理都需要编写不同的程序，可重用性不好。

## 5.2 结构化分析概述

这一节先讨论结构化分析，包括系统分析的任务、结构化分析的方法、结构化分析的工具及系统要求的确定等。

### 5.2.1 系统分析的任务

系统分析的重点是在应用需求层次上，主要任务是确定用户对系统的应用要求。第一，通过深入调查分析，与用户一起充分了解现行系统是怎样工作的，理解用户对现行系统的改进要求和对新系统的要求；第二，把与用户共同理解的新系统用恰当的

逻辑模型表达出来。可以说，系统分析的任务就是理解和表达。

概括地说，系统分析阶段的任务就是从逻辑上分析现行系统并设计出新的系统。其主要工作内容是：

(1) 通过对现行系统中数据和信息的流程及系统的功能给出逻辑的描述，得出现行系统的逻辑模型；

(2) 在调查和分析中得出新系统的功能需求，并给出明确的描述，得出新系统功能需求的逻辑描述；

(3) 根据需要与实现的可能性，确定新系统的功能，即给出系统功能的逻辑描述，进而得出系统的逻辑模型。

对一个较复杂的系统进行分析时，不仅要对整个系统或主系统进行分析，也要按职能/性能分成若干个较小的分系统或子系统而逐个进行分析。如果对系统分析时只考虑主系统，则分析得太粗，系统的整体轮廓不易描述清楚。但分析得太细，层次必然增多，也会使系统的轮廓变得模糊，从而体现不出系统的要点与框架。因此，必须合理地选择系统分析工作的层次。

在信息系统的开发过程中，系统分析是最困难的一部分工作，这主要体现在用户的需求如何恰当地反映到新系统中去。系统分析员就是这一重要工作的承担者，系统分析员在用户与系统设计人员之间起着桥梁与翻译的作用。系统分析员一方面要熟悉有关用户的实际业务知识，了解用户的想法和要求；另一方面要充分运用自己具有的信息系统及计算机专业知识，将用户的需求结合具体的环境，综合出一个明确的概念，即系统应具有的逻辑功能。系统分析员必须用适当的方法给出一个总体描述，使得用户与设计人员能够在这个方案上取得一致的意见。

系统分析员应充分代表用户的利益，并在整个系统开发中起作用；同时还应将用户的想法与需求限定在一个合理的范围之内。系统分析员必须协调好用户与设计人员之间的关系，使用户的合理需求能够尽可能地采用较好的技术体现在新系统中。系统分析员还必须牢牢记住：不能满足用户需求的系统，哪怕设计思想及采用技术再先进，也是一个失败的系统。因此，合理地选用先进的信息处理技术，并与用户的需求紧密结合起来，是开发成功的信息系统的决定性因素。

系统分析员一般应履行下列职责：作为用户与设计人员的接口；从各种来源收集数据，并综合出解决问题的方法；评价现行的系统，并分析新系统；不断吸收先进的科学技术，并合理地加以应用；整理、起草文档。上面这些职责结合了技巧和科学方法，系统分析员应在实践中逐步加以体会和改进自己的工作。特别强调的是，系统分析员应时刻注意倾听各方面人员的意见，但又不依赖别人的意见，而是根据客观事实来作决策，并能合乎逻辑地、抽象地、创造性地思考、综合。

一般，系统分析员在系统分析工作中易存在以下问题，必须注意解决。

(1) 由于系统分析员缺少足够的业务知识，对用户很难达到完全的理解，因而确定出来的新系统的逻辑功能可能难以满足用户的要求。

(2) 系统分析员所面对的用户可能对计算机和信息系统缺乏足够的了解，不知道在新的环境中如何反映自己的要求，即不知道哪些工作计算机能够做到，而哪些工作计算机又不能做到。

(3) 在实地调查中，涉及大量的报表、资料和数字，系统分析员很容易被埋没在这些具体的资料之中，得不出一个综合的概念，从而“只见树木不见森林”。

(4) 系统分析员编写的系统分析资料要么充满计算机术语和符号，使用户难以理解；要么专为用户编写，缺乏科学性和严谨性，使系统设计人员难以开展下一步的工作，即存在着一种“沟通”上的障碍。

### 5.2.2 结构化分析的方法

随着信息系统规模和计算机应用领域的不断扩大，不规范的、手工作坊式的分析方法已越来越不适应发展的需要，对于系统分析员来说，迫切需要采用一种规范的方法和工具克服在系统分析中面临的困难，更好地沟通用户与设计人员的理解，做好系统分析工作。

20 世纪 60 年代末提出的结构程序设计方法对信息系统的分析与设计产生了深刻的影响，启发了人们的思路。既然一个程序可以用一组标准和方法加以构造，那么在系统分析与设计工作中，也可以引进一组标准的准则和工具，来帮助人们分析、设计一个信息系统。结构化分析方法就是用一组标准的准则和工具，从事系统分析工作，并用来表达系统分析的工作成果的。该方法的基本思想和特点如下所述。

(1) 采用一组图形化工具作为通信媒介，沟通用户与设计人员的理解，并通过这组图形反映整个信息系统的信息流向、信息存储和具体功能的描述等。

(2) 采用自顶向下的方法，把系统需求和功能按层次地分解、描述并兼顾系统的测试。对用户来说，不但能对系统有一个总的概念性的印象，而且可以了解具体的组成部分，能够尽可能早地展望结果，及时提出意见。对设计人员来说，不但可更清楚地了解系统，也可相应地用自顶向下的方法来设计系统。

(3) 吸收用户参与系统的开发，阐述新系统能够“做什么”，而不是关心“怎么做”，即强调系统的逻辑模型而不是物理模型。

(4) 编制既适用于用户，又适用于系统设计人员的系统分析资料，这样既避免了重复性工作，也增强了一致性和可修改性。

### 5.2.3 结构化分析的工具

结构化分析的常用工具有以下几种：

- (1) 数据流图；
- (2) 数据字典；
- (3) 数据存储规范化；
- (4) 数据立即存取图；
- (5) 功能分析的表达方法，包括决策树、决策表和结构式语言。

其中前四种工具主要是针对数据分析，第五种工具为功能分析工具，它们包括了系统分析的基本工作，可以帮助系统分析员更好地从事系统分析工作，起好桥梁与翻译的作用。此外，另一种结构化分析工具 IDEF0 也常用来对系统进行需求分析。

数据流图用以表达系统的数据来源和走向，并指出系统中的各逻辑功能及联结方式。数据字典详细定义了数据流图中的所有数据结构及数据流，是数据流图的重要补充。在结构化系统分析中，数据流图与数据字典是最主要的工具。数据流图中每一个逻辑功能都需要用自顶向下法予以分解，并通过决策树、决策表及结构式语言等对处理过程加以描述。数据流图中的每一个数据存储要力求简单规范，以保证数据的一致性。对特定的数据存取要求，要用数据立即存取图加以表达。图 5.2 简单描述了这些工具之间的关系。

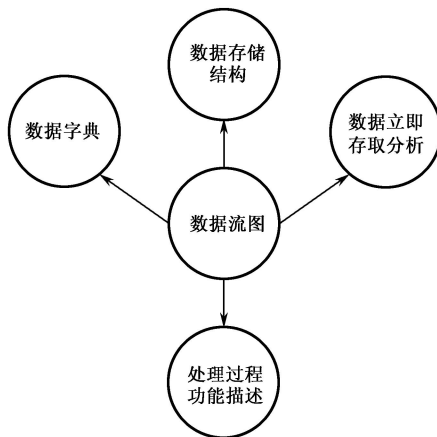


图 5.2 结构化分析中各种工具之间的关系

5.2.4 系统要求的确定

系统分析工作的核心是在现行系统的基础上，确定新系统的逻辑模型，这就要求系统分析员要对具体业务领域内的所有重要方面进行深入细致的调查，从而了解现行系统的工作情况，以及确定用户对新的或拟改进的系统的要求。这项工作叫做系统要求的确定。

系统要求的确定主要就是研究现行系统是怎样工作的，以及怎样进行改进。在这一阶段，系统分析人员必须尽快地熟悉和了解用户领域的工作情况，运用诸如面谈、分发调查表、现场检查记录及观察等实地调查技术，掌握现行系统的真实情况，了解用户的新要求和问题所在。在这些基础上，系统分析人员必须结合自己在信息系统及计算机技术方面的知识，确定出系统的逻辑模型。

要求是新系统必须包含的一个特征，包括接收信息、处理信息进而产生信息，以控制、支持各管理部门的活动。确定要求意味着研究现有的系统和收集有关系统的细节，从而找出这些要求是什么。一般情况下，这些具体的细节总是围绕着现行系统的目标、主要功能而发挥作用的，并与现行系统的组织结构有密切的联系。因此，系统要求的确定并不是单纯地局限于现行系统的工作流程和事务流程，还与现行系统的目标、功能、组织结构等发生着关系。通常，系统分析人员必须围绕以下几个问题展开自己的调查。

- (1) 什么是基本的业务过程？
- (2) 在这些业务过程中，使用或产生什么样的信息？
- (3) 处理信息的工作量和处理时间有什么要求？
- (4) 使用什么样的性能控制？

正确地回答上述问题将有助于系统分析人员理解现行系统的完整概貌，对于进一步的分析工作具有至关重要的意义。当然，这四个问题只是笼统地说明了调查工作的各个方面，详细的调查内容见表 5.1。

表 5.1 系统要求的确定的基本内容

类 别	调 查 内 容
业务过程	组成活动的各个过程、步骤和功能；活动如何触发；与外界信息发生的联系；确定的管理目标；处理时间及可能产生的延迟；操作费用等
数据	系统所需数据的来源；系统数据的接收和存储方式；系统产生信息的使用者及所使用的数 据项；可能丢失的数据；数据的编码等
处理数量与处理时间	信息处理量多大；发生的活动量；发生周期；处理时间的要求等
性能控制	现行使用的控制方法；测量、评价性能的标准；错误检测和防范措施等
其他	系统的组织结构；影响系统的环境因素等

任何信息系统总可被理解为“输入—处理—输出”的形式，根据这一理解和表 5.1 中的有关调查内容，下面分别介绍输入信息、过程处理、输出信息以及信息编码的调查内容。

### 1. 输入信息

现行的所有输入信息要全部调查清楚。输入信息的调查对系统分析和设计有很大的影响。对每个要输入的信息组，需调查的具体事项应该有：

- (1) 输入信息组的名称；
- (2) 输入目的和使用场合；
- (3) 采集手段（人工或自动）；
- (4) 输入周期、时间；
- (5) 最大输入量、平均输入量；
- (6) 复制份数，送到何处；
- (7) 保存期限；
- (8) 产生输入信息组的部门及人员；
- (9) 数据项、位数、类型、上下界的值等。

### 2. 处理过程

这是对输入信息经过哪些处理过程变换成输出信息的调查。对每一处理过程须调查的具体事项有：

- (1) 处理加工的内容；
- (2) 处理过程名称；
- (3) 过程处理的部门；
- (4) 过程处理采用的方法、算法；
- (5) 过程处理的时间；
- (6) 产生的输出信息；
- (7) 处理时采用的核对检查措施；
- (8) 该过程处理的必要性如何；

(9) 对异常情况有无处理措施，如对异常情况有处理时，应进一步了解其处理方法、处理负责人、发生的频率、处理所需时间等。

### 3. 输出信息

系统总是为了获取输出信息而建立的，因此对现行系统的输出信息也要做详细的、不遗漏的调查，在此基础上改进并建立新系统。对每个输出信息组，要调查：

- (1) 输出信息组的名称；
- (2) 使用部门或使用者的；
- (3) 使用目的及必要性；
- (4) 产生输出信息的部门；
- (5) 产生输出信息的方法；
- (6) 制作时间和周期；
- (7) 发行份数；
- (8) 处理的信息量；
- (9) 送交方法；
- (10) 数据项名、位数、数据类型；
- (11) 核对方法；
- (12) 有关的输入信息等。

#### 4. 信息编码

编码的具体方法、形式、使用范围与使用编码的方法、方式有密切关系。现行编码（如职工编码，产品编码，部件、零件编码，物品编码等）对新系统的设计具有影响，因此对现行编码须做一番调查，调查内容如下：

- (1) 编码的名称；
- (2) 编码的方法、规则、要领；
- (3) 编码序号的总数；
- (4) 编码的位数、段数；
- (5) 起始码、最大码；
- (6) 缺码率；
- (7) 追加或作废频率；
- (8) 管理部门等。

上述四类问题的调查结果要有条理地整理成报告、报表和说明书等，这是系统分析员的职责。

新系统的模型用来刻画系统所涉及的信息、处理功能及实际运行时的外部行为。但是，系统分析阶段所建造的模型不应涉及系统的实现细节，因为这样会分散分析人员的注意力，限制系统设计人员为提高系统的质量和效率而选择实现方法的自由度。因此，系统分析阶段建立的是信息系统的逻辑模型。

通常，系统分析人员选定数据流图表示数据流、处理过程和系统行为，并利用树、表或受限的自然语言等给出用户需求的描述。



建立系统模型是分析活动的焦点。因为模型以一种简洁、准确、结构清晰的方式系统地描述了目标系统的需求，从而便于分析人员规范用户描述中的模糊性和不一致性，并使系统需求趋于完整。分析过程实际上是系统模型的建造和不断完善的过程。在分析过程的初期，系统分析人员通过访谈、会议和实地观察为构筑模型收集素材，同时也利用不太完整的初步模型作为与用户相互沟通的需求表示机制。此后，分析人员利用结构化分析方法不断地对模型进行精确化、一致化、完全化方面的工作。最终确立的系统模型——逻辑模型，是建立系统物理模型的基础，也就是系统设计和实现的基础。

## 5.3 数据流分析技术

数据流分析（Data Flow Analysis, DFA）方法源于结构化分析，是一种以数据流技术为基础的、自顶向下的、逐步求精的系统分析方法。通常所说的结构化分析就是指数据流分析。

### 5.3.1 数据流分析

数据流分析的核心特征是“分解”和“抽象”。所谓分解，是指将一个复杂的问题按照其内在的逻辑划分为若干个相对独立的子问题，从而简化复杂问题的处理。所谓抽象，就是将一些具有某些相似性质的事物的公共之处概括出来，暂时忽略其不同之处，或者说，抽象是抽象出事物的本质特性而暂时不考虑它们的细节。

例如，在图 5.3 中，系统 S 被分解为 S1、S2、S3 三个子系统，子系统 S1、S2、S3 又被分解为 S11、S12、S13，S21、S22，S31、S32。如果第二层的子系统仍然比较复杂，则还可以再进一步分解，如此下去，直到每个子系统足够简单、能清楚地被理解和表达为止。

分解和抽象实质上是一对相互有机联系的概念。在图 5.3 中，自顶向下的过程，即从顶层到第一层再到第二层的过程，称为“分解”；自底向上的过程，即从第二层到第一层再到顶层的过程，称为“抽象”。也就是说，下层是上层的分解，上层是下层的抽象。这种层次分解使我们不至于一下子考虑过多的细节，而是逐步地去了解更多的细节。如在图 5.3 中，对于顶层一般不考虑任何细节，只考虑系统对外部的输入和输出，然后一层层去了解系统内部的情况。

对任何复杂的系统，分析工作都可以按照上述方式有计划、有步骤地进行，规模大小不同的系统只是分解的层次不同，即规模大的系统分解的层次较多，规模小的系统分解的层次较少。

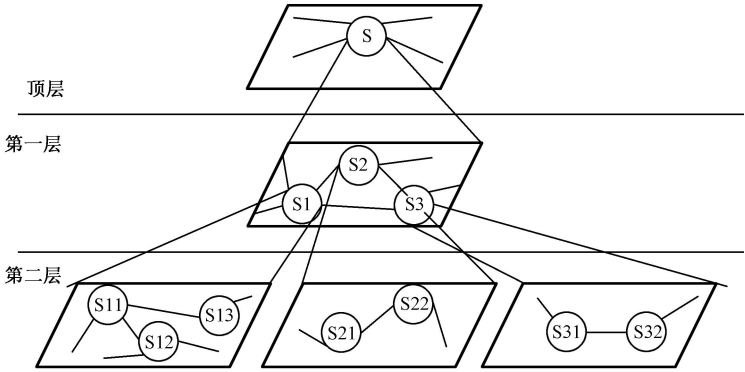


图 5.3 数据流分解示意图

结构化方法是进行可行性研究和系统分析的行之有效的方法。在结构化方法中，采用数据流图来建立系统的逻辑模型，采用数据字典对数据流图进行说明，采用决策树、决策表和结构式语言等对处理过程进行描述。

5.3.2 数据流图

数据流图（Data Flow Diagram，DFD）是结构化系统分析的主要工具，它能图形化地显示出系统中数据的流转和使用，表达数据在系统内部的逻辑流向，以及系统的逻辑功能和数据的逻辑变换。

下面首先介绍数据流图的画法、使用，以及如何表示数据和它的迁移过程。

数据流图有四种基本符号：外部项、数据流、处理过程和数据存储。数据流图的表示方法有多种，图 5.4 给出了三种常见的符号表示（每列为一组），本书后面主要采用第一组表示方法。

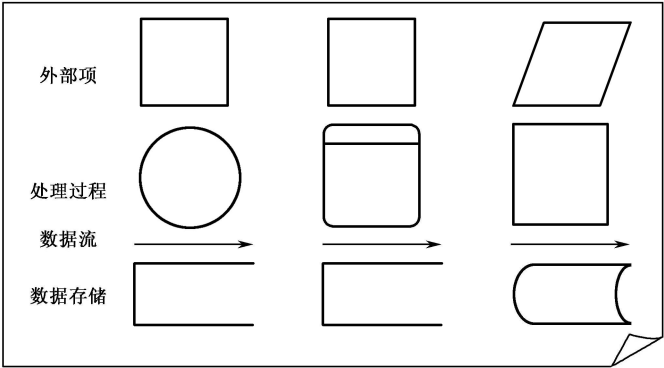


图 5.4 数据流图的表示方法

## 1. 外部项

外部项 (External Entity) 是指不受系统控制的, 在系统以外的人、程序、机构或其他实体, 外部项与系统通过数据交互, 表达了对应数据的外部来源或去处。例如, 学生、顾客、车间、财务部门、经理等都可能作为外部项。外部项也可以是一个向本系统提供数据或接收本系统输出数据的另外一个信息系统。外部项用一个矩形来表示, 矩形内一般填写该外部项的名称, 以区别于其他外部项。如果一个外部项在同一张图上出现多次, 则可在其角上画上一小斜线, 表示重复项或副本。

确定系统的外部项, 实际上就是确定系统与外界的分界线。系统与外界的境界必须在详细地分析用户的需要及系统目标的基础上加以确定。

## 2. 数据流

数据流 (Data Flow) 就是一束按特定的方向从源点流到终点的数据, 它指出了数据及其流动方向。一般用一条线表示数据流, 用箭头指示流动方向。数据流可以由某一外部项产生, 也可以由处理过程或数据存储产生。对每一条数据流都要命名, 并标识在数据流箭头的上方, 以便使用户和系统设计人员能够理解它的含义。

有时候, 很难用简单而又恰当的名词来表达某一数据流的内容, 这时可用带定语的名词短语来命名。

例如, 某工厂的顾客可能寄来订货单、支票、退货单, 或是来询问某件事, 处理后有相应的输出产生, 把这些数据流一一表示出来, 数据流图就如图 5.5 所示。

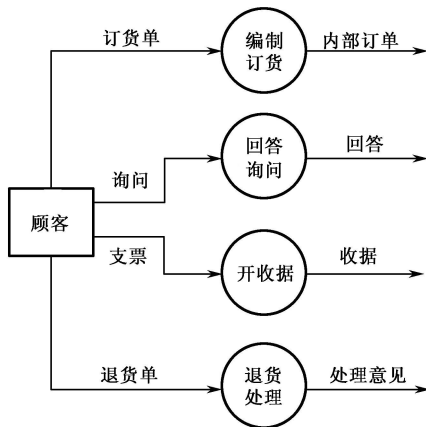


图 5.5 数据流图示例

### 3. 处理过程

处理过程（Process）是对数据进行变换的操作，即把流向它的数据流进行一定的变换处理，产生新的数据流。通常用圆圈表示一个处理过程，如图 5.5 中出现的“退货处理”等。处理过程的名字填写在圆圈内，并应适当反映该处理的含义，使之容易理解，一般是动词短语。还应给每个处理过程一个编号，具有层次的编号可以说明该处理过程在层次分解中的位置。

处理过程对数据的操作主要有两类：

- （1）变换数据的结构，如将数据的格式重新排列；
- （2）在原有数据内容的基础上产生新的数据内容，如对数据进行累计或计算平均值等数学运算。

### 4. 数据存储

数据流仅表达数据的流动方向，数据的保存则由数据存储来表达。数据存储（Data Store）指出了数据保存的地方，这里所说的地方，并不指保存数据的物理地点或物理存储介质，而是数据存储的逻辑描述。数据存储可用右边开口的长方形来表示，长方形内部填写该数据存储的名称。

为避免数据流线条的交叉，有时在一张图中会出现同样名称的数据存储。此时可在重复出现的数据存储符号的左边再加一条或两条竖线或在左上角画一条或两条斜线。有时为区别于其他数据存储，除了名称外，再给数据存储一个编号标识，可由英文字母 D 和数字组成。

## 5.3.3 数据流图的建立

对一个大而复杂的系统来说，如果分析人员试图一次就建立它的数据流图，并且将所有的外部项、处理过程、数据存储等画在一张图上，那只能导致失败，系统中大量的数据流和处理过程将似一团乱麻，甚至把分析人员自己也搞糊涂了。因此，在建立数据流图时，应掌握一定的方法和技巧。

### 1. 自顶向下扩展

为了画出正确的数据流图，通常采用的方法是自顶向下扩展，这是一个自然且较有条理的思考过程。自顶向下扩展方法是，先用少数几个处理过程高度概括、抽象地描述整个系统的逻辑功能，然后针对处理过程逐步地分解、扩展，从而详细地加以描述。

数据流图可在不同的抽象层次上描述，以表示系统在该层的内容。最高层次的数据流图（可称为背景图）反映整个系统的概貌，它确定目标系统与外部环境的关系及系统的来龙去脉或范围。无论高层还是其他层次的数据流图，其建立方法一般都是遵循如下顺序：

(1) 决定系统或处理过程的范围，即通过外部项，输入/输出数据流确定系统的边界或处理过程的范围；

(2) 决定系统或处理过程内部的细节，并加以描述。

下面以一个例子来说明数据流图的建立过程。图 5.6 所示是某中心城市的自来水收费系统的背景图。整个收费系统用图中央的一个圆圈表示，该图简要地显示了系统本身和外部实体间的数据和信息流向。

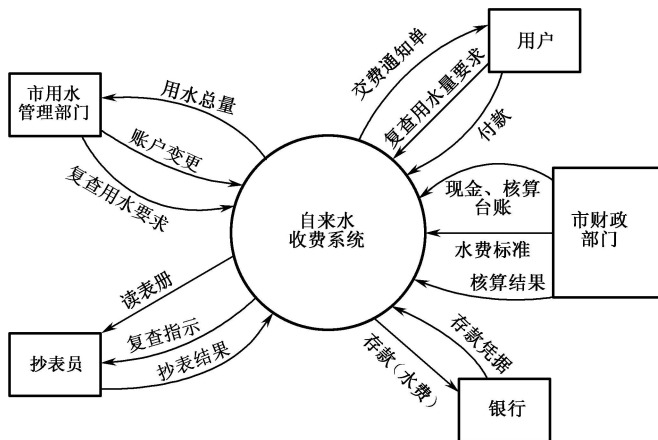


图 5.6 某自来水收费系统的背景图

该自来水收费系统由市政府经营并管理。读水表之前，由收费系统定期（如每户一月一次）给抄表员该录回的用户用水量记录卡（读表册），抄表员根据读卡进行读录，并将读录结果输送到系统。系统给用户发出缴纳水费通知单。若用户认为抄表员读录的用水量合理，则去银行付款；若认为不合理则可提出复查要求。对于请示复查的用户，系统将给抄表员发出复查该户水表的指示。系统还将定期地向市用水管理部门报告公共设施用水总量，用水管理部门必要时可向系统提出复查。当用户发生变化时，用水管理部门将通知系统有关变更情况。收缴水费所得款项上报市财政部门。系统委托银行收款，用户向银行交款，银行向系统递交存款凭据。

顶层数据流图简明地把系统和外部环境的关系刻画在一张图表上，图中标出的数据流均限于系统中重要的信息，如读表册、缴费通知单等。



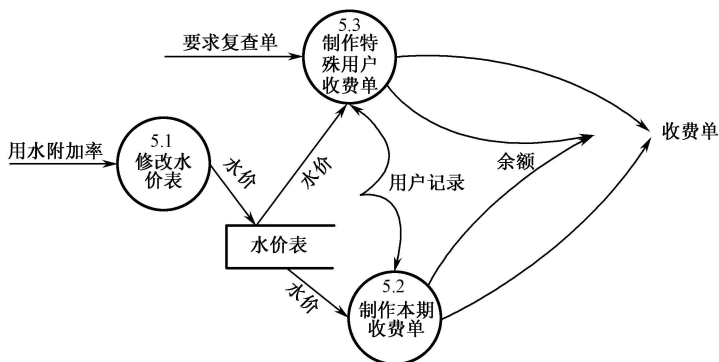


图 5.8 处理过程 5 的展开

但是，图 5.8 中的几个处理过程仍然不够详细，还需继续分解、展开。例如，“制作本期收费单”部分，可进一步扩展为图 5.9 所示的形式。

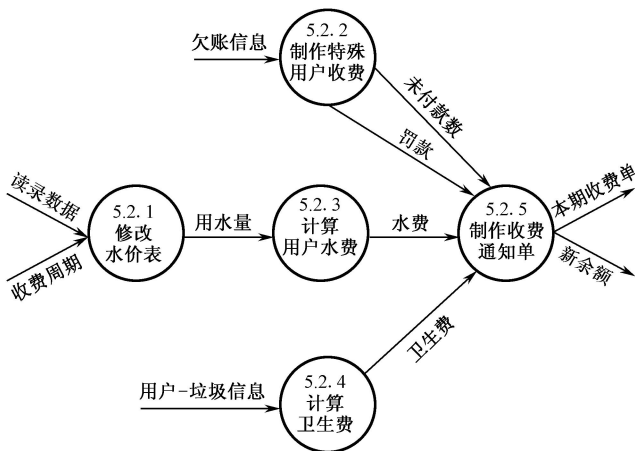


图 5.9 处理过程 5.2 的展开

由图 5.9 可以看出，处理过程的序号是在上层过程的序号上继续增加的。图中列出了计算收费的细节：正常的用水费、拖欠缴纳用水费的罚款及清除居民垃圾的卫生费。本系统是市政管理的一个组成部分，征收居民清理垃圾的卫生费是随水费交款通知单一并开出的，其征收标准随居民区的不同而发生变化。

限于篇幅，仅给出部分处理过程的展开示例。所有需分解的过程都应作同样的展开处理，并逐级画出其数据流图，直至最低一层的过程可用简短的说明描述清楚为止。

## 2. 建立数据流图的原则

数据流图的建立过程必须遵循自顶向下、逐层分解的原则，这是控制系统复杂性的重要方法，也是结构化系统分析的基础。逐层分解的方式不是一下子引进太多细节，而是有控制地逐步增加细节，实现从抽象到具体的逐步过渡，因而将有利于对问题的理解。

用自顶向下、逐层分解的原则来画数据流图，就得到了一套分层的数据流图，分层的数据流图总是由顶层、中间层和底层组成的。顶层数据流图确定了系统的边界，这是工作的第一步。为了画出顶层数据流图，必须首先识别不受系统控制的，但影响系统运行的外部因素，从而确定出系统的外部项和系统的数据输入源和输出对象。通过对顶层图的了解、展开，将得到许多中间层的数据流图。中层图描述了某个处理过程的分解，而其组成部分又要进一步被分解。一个大型系统的中间层可能有七八层之多。中间层的展开应是化复杂为简单，但决不能失去原有的特性、功能和目标，而应始终保持系统的完整性和一致性。如果展开出来的数据流图已经基本表达了系统所有的逻辑功能和其必要的输入、输出，处理过程已经足够简单，不必再分解时，则即得到了底层数据流图。底层数据流图所描述的都是无须分解的基本处理过程。

系统分析人员在分解数据流图的过程中，应时刻注意以下两点：所画出的数据流图是否简单、明确，能否既被用户理解又使系统设计人员明白；是否既能描述现行系统的逻辑功能，又能反映将要建立的新系统的逻辑功能。

建立正确的数据流图要求系统分析人员不断地实践、总结，逐步理解和掌握有关的方法和经验。下面所列出的内容对于系统分析人员画出正确的数据流图具有一定的指导意义。

(1) 确定系统的外部项及系统正常运行时的输入与输出，在高层的数据流图中只反映主要的、正常的逻辑功能，突出系统的总体情况。

(2) 由外向里、从左到右地画数据流图，先在左侧画外部项，然后画出由该外部项产生的数据流和其对应的处理过程。接收系统数据的外部项一般画在数据流图的右侧。

(3) 适当地命名及给出编号，有利于系统的理解。对处理过程的编号，随着逐层展开，也应反映出它的层次关系。例如，第一层图中处理过程的编号为 1, 2, …；第二层图的编号应是 1.1, 1.2, …, 2.1, 2.2, …；依此方法，逐层给处理过程加上层次的序列号。

(4) 应集中精力于主要的数据流，对一些诸如例外情况、出错处理等问题不必花较多的精力分析下去，有时只须标出即可。



(5) 一个数据流图所包含的处理过程应限在七个以内, 经验证明, 多于七个将会影响阅读和理解效果。

(6) 数据流图逐层分解时, 应在概念上合理、清晰, 分解要自然, 不影响图的易理解性。合理的分解是将一个问题分成相对独立的几个部分, 并尽量保证其相对独立性, 减少相互之间的联系。分解应力求均匀, 避免在同一张数据流图中, 有些处理过程描述的是细节, 而另一些描述的却是较高层的抽象。

(7) 数据流图与程序流程图(框图)不同。前者不反映时间的顺序, 只反映数据的流向、逻辑处理和必要的逻辑数据存储; 后者有严格的时间顺序, 有起始点和终止点。

(8) 数据流图不反映判断和控制条件, 不应在数据流图上出现表明控制逻辑的数据流, 判断和控制是处理过程内部的事。

(9) 理解一个问题总要经过从不正确到正确、从不恰当到恰当的过程, 系统分析人员要随时准备修改甚至抛弃旧的数据流图, 而用更好的来替代。分析阶段重画几张图的代价是小的, 倘若草草了事, 留下隐患, 那么到开发后期再去纠正, 代价就太大了。因此, 系统分析人员要有足够的耐心, 花大力气去了解、分析系统, 建立正确、完整的数据流图。

### 3. 数据流图的一些绘图规则

除了上面的原则外, 还有一些数据流图的绘图规则, 可以帮助提高绘图的正确性, 并可直观地从图上排除一些简单的错误。

(1) 处理过程。处理过程不能只有输出, 或只有输入, 而且输入和输出不能完全相同。

(2) 外部项。数据流不能直接从一个外部项到另一个外部项, 而必须通过处理过程。

(3) 数据存储。数据流不能直接从一个数据存储到另一个数据存储, 而必须通过处理过程。数据流不能直接从一个外部项到一个数据存储, 反之也不行, 而必须通过处理过程。

(4) 数据流。数据流只能单向。数据流不能直接流入它所流出的处理过程, 而必须通过其他处理过程, 产生新数据流的同时, 将原数据流返回。数据流进入数据存储表示更新, 离开数据存储表示检索。

### 5.3.4 数据字典

数据流图是结构化系统分析中不可缺少的有力工具, 它描述了系统的分解, 即系

统由哪些部分组成，各部分之间有什么联系等。但是，它还不能完整地表达一个系统的全部逻辑特征，特别是有关数据的详细内容。因此，仅仅一套数据流图并不能构成系统说明书，只有当图中出现的每一个成分都给出详细定义之后，才能较全面地描述一个系统。

数据流图中所有名字的定义及描述就构成了一本字典，它包括数据流、数据存储、外部项和处理过程的详细条目。数据流、数据存储等数据型条目构成数据字典（Data Dictionary），而逻辑分析的有关工具用于处理型条目。

数据字典把数据流图上的所有数据都加以定义，并按特定格式予以记录，以备随时查询和修改。因此，数据字典是数据流图的辅助资料，对数据流图起注解作用。在结构化系统分析中，数据字典主要用于描述数据流和数据存储的逻辑内容，以及外部项和处理过程的某些数据特性。

数据字典中把数据的最小组成单位定义为数据项，而若干个数据项可以组成一个数据结构。数据字典是通过以数据项和数据结构的定义来描述数据流、数据存储的逻辑内容的。

## 1. 数据项

数据项是数据的最小组成单位，即不可再分的数据单位。例如，学生的“姓名”可以看作是一个数据项，但要注意此时“姓”和“名”不能分开表示，如果分开表示，那么“姓名”就不是数据项了。

在数据字典中，数据项的定义有以下内容。

（1）数据项的名称：每个数据项均有一个名称加以标识。例如，职工号、职工名、产品名、考核成绩等都是数据项的名称。在整个系统中，数据项的名称应唯一地标识出这个数据项，以区别于其他数据项。数据项的名称应尽量反映该数据项的具体含义，以便容易理解和记忆。同一数据项，其名称可能不止一个，以适于多种场合下的应用。在这种情况下，还需对数据项的别名加以说明。

（2）数据项的值域：指数据项的取值范围及每一个值的确切含义。例如，某企业职工“工资”的值域是800~5 000元之间的数值。又如，人事档案中的“文化程度”数据项，如果规定只能取“小学”、“初中”、“高中”、“中专”、“大专”、“本科”、“研究生”这七个值中的任意一个，则“文化程度”这一数据项的值域就是上述所列的七个值。如果用字母或缩写代替数据项的值，还需说明字母或缩写的含义，亦即说明数据项的取值含义。数据字典中应对每一个数据项的值域和取值含义都加以定义，以便在分析问题时加以使用。数据项的值域对于输入数据项时的检、纠错起着重要的作用。

（3）数据项的数据类型：指取值的数据类型。基本类型有数值型（包括整数与实

数)、字符型(包括汉字的使用)、逻辑型等。例如,职工“工资”的数据项为数值型,“文化程度”为字符型。

(4) 数据项的长度:规定该数据项所占的字符或数字的个数。例如,“文化程度”数据项的长度为 6 位(3 个汉字所占的字符长度)。

除了上述 4 项主要内容外,必要时还须对数据项的简单描述、与之相关的数据项或数据结构、处理过程等加以说明。

## 2. 数据结构

数据结构用来定义数据项之间的组合关系。数据字典中的数据结构是对数据的一种逻辑描述,与物理实现无关。一个数据结构可以是若干个数据项的组合,也可以由若干个数据结构组成,还可以由若干个数据项和数据结构混合组成。

在数据字典中,对数据结构的定义包括数据结构的名称和数据结构的组成。

(1) 数据结构的名称:用于唯一标识这个数据结构,以区别于系统中其他的数据结构,如“职工工资单”、“学生档案”等。

(2) 数据结构的组成:包括数据项或数据结构。如果引用了其他数据结构,那么被引用的数据结构应已被定义,而且这里只须列出被引用的数据结构的名称。

对数据结构的定义还包括数据结构的简单描述、与之相关的数据流、数据结构或处理过程及该数据结构可能的组织方式。

下面给出一个数据结构定义示例。

数据结构名称:课程

简述:用于记录有关课程安排的基本信息

组成:课程名

教师

教材

课程表

组织:按课程名顺序存放

有关的数据流 数据结构:教师档案、教材表

有关的过程:排定课程表、教学查询

上例中,“课程表”是一个数据结构,它是由授课时间(“星期几”、“第几节”)及教室组成的。有关“课程表”数据结构的定义应当在数据字典中可以查到。

## 3. 数据流

数据流表明数据项或数据结构在系统内传输的路径。在数据字典中,对数据流的

定义包括以下几点。

(1) 数据流的来源：数据流的源点，它可能来自系统的外部项，也可能来自某一个处理过程或一个数据存储单元。

(2) 数据流的去向：数据流的终点，它可能终止于外部项、处理过程或数据存储。

(3) 数据流的组成：所包含的数据项或数据结构。一个数据流可能包含若干个数据结构，这时需在数据字典中加以定义。如果一个数据流仅包含一个简单的数据项或数据结构，则该数据流无须专门定义，只需在数据项或数据结构的定义中加以标明。

(4) 数据流的流量：在单位时间内，该数据流的传输次数，如 500 次/天。

(5) 高峰时的流量。

下面给出一个数据流定义示例。

数据流名称：提货单

简述：生产科向库房发出的提货单据

数据流来源：生产科

数据流去向：“提货处理”过程

数据流组成：提货单号

提货单标识

提货者

品种

数量

流量：份 天

高峰流量：月初时，份 天

#### 4. 数据存储

数据存储指数据结构暂时或被永久保存的地方。在数据字典中，只能对数据存储从逻辑上加以简单的描述，不涉及具体的设计和组织。通常，定义数据存储的内容有：

(1) 数据存储的名称及必要时所给的编号；

(2) 流入/流出的数据流；

(3) 数据存储的组成，即它所包含的数据结构；

(4) 存取分析及关键字说明等。

下面给出一个数据存储定义示例。

数据存储名称：学生成绩

编号：

简述：记录学生所考各门课程的考试成绩

流入的数据流：“考试成绩单”，来源是“登记成绩单”处理过程

流出的数据流：“成绩”，去向是“成绩统计”处理过程

数据存储的组成：学号

课程号

成绩

## 5. 处理过程

对处理过程中具体操作的描述，不属于数据字典的范围，在“逻辑分析工具”一节中，将作专门的介绍。这里仅对处理过程的部分数据特性作简单的描述，以便从数据字典中能得到系统所有部分的说明，利于检索和查对。在数据字典中，对处理过程的描述有以下几项内容：

- (1) 处理过程在数据流图中的名称、编号；
- (2) 对处理过程的简单描述；
- (3) 该处理过程的输入数据流、输出数据流及其来源与去向；
- (4) 其主要功能的简单描述。

下面给出一个处理过程定义示例。

处理过程名称：编辑学生成绩单

编号： . .

简述：将学生某门课程的考试成绩录入系统，并产生某学生已考课程的成绩单

输入：课程成绩单，来源为外部项“教师”

处理：按一定的格式将所有学生该门课程的考试成绩一次录入“考试成绩”数据存储；

根据学号将该课程的考试成绩分别转入该学号的“成绩单”数据存储

输出：考试成绩，去向为“成绩单”数据存储、“确定补考”处理过程

## 6. 外部项

在数据字典中，对外部项的定义包括：

- (1) 外部项的名称；
- (2) 对外部项的简述；
- (3) 有关的数据流。

一个信息系统的外部项不应过多，否则系统的独立性不好。此时，需重新考虑系统界面，设法减少外部项。

下面给出一个外部项定义示例。

外部项名称：财务处  
简述：处理企业内部财务工作的职能部门  
有关的数据流：工资单、成本、利润等

上述六个方面的定义构成了数据字典的全部内容，在实际应用中，常常将数据存储和处理过程的描述另立报告，而不在数据字典中描述；另外，有时也可省去一些内容，如外部项的描述。但是，数据项、数据结构和数据流必须列入数据字典中，加以详细说明。

数据字典的内容是随着数据流图自顶向下、逐层扩展而不断充实的。数据流图的修改与完善，将导致数据字典的修改，这样才能保持数据字典的一致性和完整性。数据字典的建立可以有两种方式，一是由人工将有关内容随时建立在一叠卡片上，对卡片进行分类、排序，从而得到数据字典；二是使用自动化数据字典系统，由计算机来代替人工登记、分类等工作。对于小规模的信息系统来说，人工建立数据字典是较为合适的，但对于中、大型的信息系统，则应建立一部自动化的数据字典，以提高工作效率。

数据字典的建立，对于系统分析人员、用户或系统设计人员均有很大的好处，他们可以从不同的角度分别从数据字典中得到有关的信息，便于认识整个系统和随时查询系统中的部分信息。随着系统开发工作的不断深入，数据字典所带来的效益也将越来越明显。

### 5.4 IDEF0 分析技术

IDEF0 也是一种结构化分析技术，来源于结构化分析设计技术方法，由图形化及结构化的方式，清楚严谨地将一个系统中的功能，以及功能之间的限制、关系、相关信息与对象表达出来。IDEF0 是 IDEF 方法族中的方法之一。

在 IDEF 方法族中，最初只有 IDEF0、IDEF1 和 IDEF2 三种方法。IDEF0 方法在 ICAM 中用来建立加工制造业的体系结构模型，以提高生产率。随着它们在实际中的广泛应用，其应用效果日益显著。目前研究机构正在努力将 IDEF 方法发展成为一个更加完整的方法系列，如表 5.2 所示。

表 5.2 IDEF 方法列表

IDEF0	Function Modeling	功能建模
IDEF1	Information Modeling	信息建模
IDEF1X	Data Modeling	数据建模

(续表)

IDEF2	Simulation Model Design	仿真模型设计
IDEF3	Process Description Capture	过程描述
IDEF4	Object-oriented Design	面向对象设计
IDEF5	Ontology Description Capture	本体论描述
IDEF6	Design Rational Capture	设计原理
IDEF7	Information System Auditing	信息系统审定
IDEF8	Human-system Interface Design User Interface Modeling	人-系统接口设计 用户接口建模
IDEF9	Business Constraint Discovery Scenario-driven IS Design	经营约束发现 场景驱动信息系统设计
IDEF10	Information Artifact Modeling Implementation Architecture Modeling	信息制品建模 实施体系结构建模
IDEF11	Information Artifact Modeling	信息工具建模
IDEF12	Organization Design Organization Modeling	组织设计 组织建模
IDEF13	Three Schema Mapping Design	三模式映射设计
IDEF14	Network Design	网络设计

在上述 IDEF 方法族中，最为成熟的是 IDEF0 和 IDEF1X。本节主要介绍 IDEF0 方法。

5.4.1 IDEF0 的特点

IDEF0 具有以下一组基本特点，这些特点形成一种思维规则，适用于从信息系统的规划阶段到设计阶段的各项工作。

1. 全面地描述系统，通过建立模型来理解一个系统

一般地说，一个系统可以被认为是由对象物体（用数据表示）和活动（由人、机器和软件来执行）及它们之间的联系组成的。IDEF0 能同时表达系统的活动（用图形盒子表示）和数据流（用箭头表示）及它们之间的联系，因此 IDEF0 模型能全面描述信息系统。

对于新的系统来说，IDEF0 能描述新系统的功能及需求，能表达一个符合需求及能完成的功能的实现。对已有系统来说，IDEF0 能分析应用系统的工作目的、完成的功能及记录实现的机制。这两种情况都是通过建立一种 IDEF0 模型来体现的。这里的模型就是系统的一种书面描述。它不一定必须用某种数学公式表示，可以是图形，甚至可以是文字叙述。因而可以说：不管何种形式，只要 M 能回答有关实际对象 A 的所

要研究的问题，就可以说 M 是 A 的模型。对于一些复杂的企业对象或其他系统，由于用自然语言无法精确又无二义性地表示分析及设计结果，所以这里采用一种图形语言来表示模型。这种图形语言能做到：

- (1) 有控制地逐步展开细节；
- (2) 精确性及准确性；
- (3) 注意模型的接口；
- (4) 提供一套强有力的分析和设计词汇。

一个模型由图形、文字说明、词汇表及相互的交叉引用表组成，其中图形是主要成分。IDEF0 图形中同时考虑活动、信息及接口条件，它把方盒作为活动，用箭头表示数据及接口。因此在表示一种当前的操作、功能说明或设计时，总是由一个活动模型、一个信息模型及一个用户接口模型组成。

## 2. 目的与观点

由于模型是一个书面说明，因此像一切技术文件一样，每一个模型都有一个目的和一个观点。目的是指建模的意义，即为什么要建立模型；观点是指从哪个角度去反映问题或者站在什么立场来分析问题。功能模型是为了进一步做好需求分析、实现预定的技术要求，所以要明确是对功能活动进行分析（逐步分解），而不是对组织机构的分解。一个活动可能由某个职能部门来完成，但活动功能不等于组织，因此必须避免画成组织模型的分解过程。模型描述的内容反映各种用户的要求，从单一角度描述问题是困难的，也是不可能的。例如，物资管理人员关心物料的收、发、存，计划人员关心什么时候物料从库存点到采购点，厂长关心哪一个工程项目节约用料，加快进度。因此要求所有的用户有同样重要而且相同的需求是不可能的、不切实际的。IDEF0 要求在画出整个系统的功能模型时，具有明确的目的与观点。譬如对一个企业的系统，必须有明确的站在厂长（或经理）的位置上建模的观点，所有不同层次的设计者都要以全局的观点来进行建模工作，这样才能保证是从全局的高度来揭示各部分之间的相互联系和相互制约的关系的。否则有的人强调设计处的利益，有的人突出供销处的要求，甚至有的可以只为某个岗位的操作人员的要求来建立各功能模块之间的联系，这样就丢失了系统性的要求。

## 3. 区别“什么（What）”和“如何（How）”

“什么”是指一个系统必须完成的是“什么”功能，“如何”是指系统为完成指定功能而应“如何”建立。也就是说，在一个模型中应能明确地区别出功能与实现间的差别。



IDEF0 首先建立功能模型。把表示“这个问题是什么”的分析阶段，与“这个问题是如何处理与实现”的设计阶段仔细地区别开来。这样在决定解法的细节之前，保证能完整而清晰地理解问题。这是系统成功开发的关键所在。

在设计阶段，要逐渐识别各种能用来实现所需功能的机制，识别选择适当机制的依据是设计经验及对性能约束的知识。根据不同模型，机制可以是很抽象的，也可以是很具体的。重要的是，机制指出了“什么”是“如何”实现的。IDEF0 提供了一种记号，来表示在功能模型中如何提供一个机制来实现一个功能及单个机制如何在功能模型的几个不同地方完成有关功能。

有时机制相当复杂，以致机制本身需要进行功能分解。

#### 4. 自顶向下分解

用严格的自顶向下、逐层分解的方式来构造模型，使其主要功能在顶层说明，然后分解得到逐层的有明确范围的细节表示。

IDEF0 在建模开始，先定义系统的内外关系、来龙去脉。用一个盒子及其接口箭头来表示，确定了系统范围，如图 5.10 所示。

由于在顶层的单个方盒代表了整个系统，所以写在方盒中的说明性短语是比较一般的、抽象的，这一点与数据流图的方法类似。同样，接口箭头代表了整个系统对外界的全部接口，所以写在箭头旁边的标记也是一般的、抽象的。然后，把这个将系统当作单一模块的盒子分解成另一张图形。这张图形上有几个盒子，盒子间用箭头连接。这就是单个父模块所相对的各个子模块。这些分解得到的子模块，也是由盒子表示的，其边界由接口箭头来确定。每一个子模块可以同样地细分，得到更详细的细节，如图 5.11 所示。

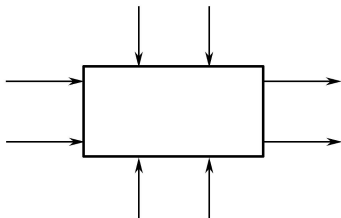


图 5.10 盒子及其接口箭头

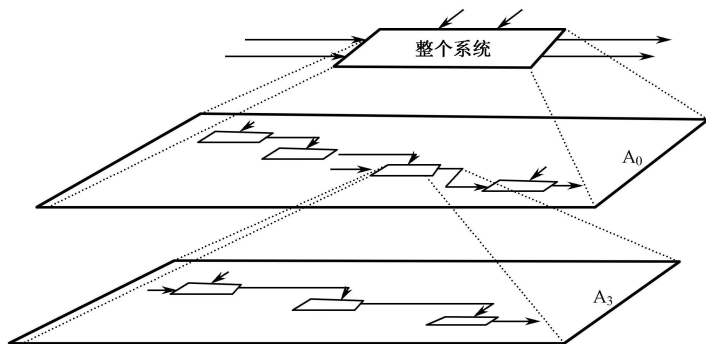


图 5.11 递阶分解结构

IDEF0 提供的规则，保证了如何通过分解得到人们所需要的具体信息。一个模块在向下分解时，通常分解成不少于 3 个、不多于 7 个的子模块。上界 7，保证了采用递阶层次来描述复杂事物时，同一层次中的模块数不会太多，以致不适宜于人的认识规律。下界 3，保证了分解是有意义的。但是，原始的 SADT 方法，规定一张图上的盒子数为 2~7 个，故通常也不作很硬性的限制。

模型中一个图形与其他图形间的关系，用互相连接的箭头来表示。当一个模块被分解成几个子模块时，用箭头表示各子模块之间的接口。每个子模块的名字加上带标签的接口，确定了一个范围，规定了子模块细节的内容。

在所有情况下，子模块忠实地代表了父模块，以既不增加也不减少的方式反映着各自父模块所包含的信息。

### 5.4.2 功能模型的表示

IDEF0 建模方法是通过一系列图形符号来表示模型的。表示的图形元素主要有盒子（Box）和箭头（Arrow）。把描述某个功能活动的图形称为活动（Activity），用盒子表示，而箭头表示系统处理的事件或数据。将系统中的所有活动及各个活动之间的处理联系相互连接起来，构成对系统的完整描述，从而形成了系统的 IDEF0 模型。

#### 1. 活动图形

按照结构化方法自顶向下、逐步求精的分析原则，IDEF0 的初始图形首先描述了系统的最一般、最抽象的特征，确定了系统的边界和功能概貌。然后，对初始图形中所包含的各个部分按照 IDEF0 方法进行逐步分解，形成对系统较为详细的描述并得到较为细化的图形表示，这样经过多次分解，最终得到的图形细致到足以描述整个系统的功能为止。

每个详细图是其较抽象图的一个分解，把较抽象图称为父图，详细图称为子图。父图中的一个盒子（活动）可以由子图中的多个盒子（活动）和箭头来进一步描述，并且父图中进入和离开的箭头必须与子图中进入和离开的箭头相一致，即父图和子图必须是平衡的。

通常在分解时，每个图形中活动的数量最好不要超过 7 个，这样做一方面可以控制模型的复杂程度，另一方面可以控制抽象的级别，同时符合人们认识问题的思维习惯。

#### 2. 盒子

IDEF0 模型是真实世界中事物功能的抽象。盒子代表活动，一般用动词短语描述，

标注在盒子的内部，并且常在盒子的下角写上编号。连到盒子上的箭头，表示由活动产生或所需要的数据等，用名词短语标注在箭头旁边。这些数据可以是信息、对象或任意用名词或短语描述的任何事物。箭头仅限制了活动间的关系，并不表示活动的顺序。活动的一般形式如图 5.12 所示。

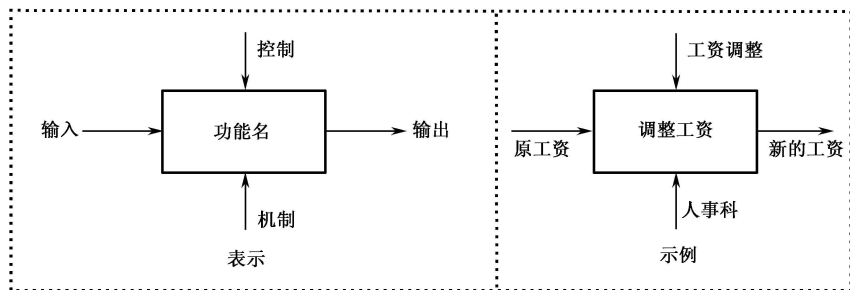


图 5.12 活动的一般形式及示例

作用于活动的箭头可以分为四类。

- (1) 输入 (Input)：功能需要处理的数据。箭头标注在活动的左边。
- (2) 输出 (Output)：功能处理得到的数据。箭头标注在活动的右边。
- (3) 控制 (Control)：说明控制变化的条件和环境（或者说约束）箭头标注在活动的上边。
- (4) 机制 (Mechanism)：作用在活动底部的是机制，它说明执行活动的事物，可以是人或设备等。

在图 5.12 中，示例表示的含义是：工资调整活动将输入的原工资调整以后得到新的工资，其中约束（控制）条件，比如增加 100 元，由人事科负责执行（机制）。

理解输入、控制两者不同的含义，对理解系统的工作是重要的。在输入与控制无法明确区分时，可将其看作控制。每个活动至少有一个控制箭头。

在活动所表示的盒子中，输入/输出箭头表示活动进行的是什么 (What)，控制箭头表示为何做 (Why)，机制箭头表示如何做 (How)，如图 5.13 所示。

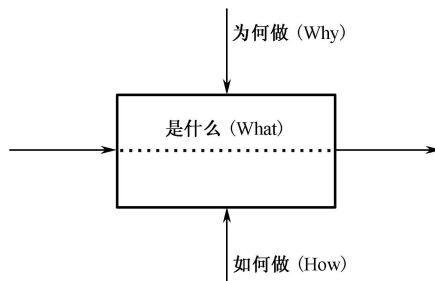


图 5.13 箭头的作用

盒子所表示的活动，往往是一组相关的活动，不一定是单一的功能。在不同条件下，作用在活动上的输入、输出、控制和机制箭头允许有多个，表示在不同的输入或控制下执行不同的功能，产生不同的输出。

3. 箭头

在活动图形上，箭头仅代表数据约束，不标明顺序和时间。一个活动的输出可以是另外一个活动的输入或控制。箭头可以联合，以表示多个活动产生同一类数据（或合成为一类数据）。箭头分支可以代表一类东西或同一种类的不同东西。

在图 5.14 中，图（a）表示活动  $A_1$  输出数据流向活动  $A_2$  和  $A_3$ ，图（b）表示活动  $A_1$ 、 $A_2$  的输出数据作为活动  $A_3$  的控制。

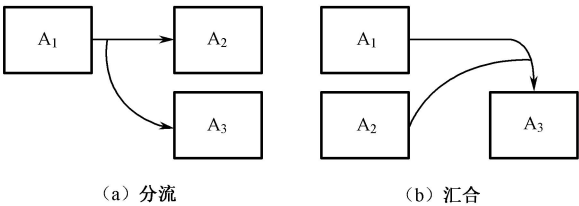


图 5.14 箭头的分流和汇合

有时在表示活动的盒子底部增加向下指出的机制箭头，称为调用（Call）箭头。它指明该活动是由什么来完成的。它已经在另一个模型中进行了细化，如需要了解细节，则可按调用箭头的图号（或节点）在另一个模型中找到有关图形，如图 5.15 所示。

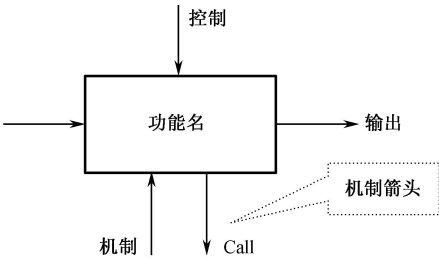


图 5.15 调用箭头的表示

如果把一个箭头在表示活动的盒子的连接端加上括号，则这样的箭头称为通道箭头，如图 5.16 所示。箭头上的括号表示该箭头将通到模型的未定义部分，与该活动的下一个子图无关，或者是众所周知，或者有共同理解的可省略的内容，在子图中为简

化图面而省略了。

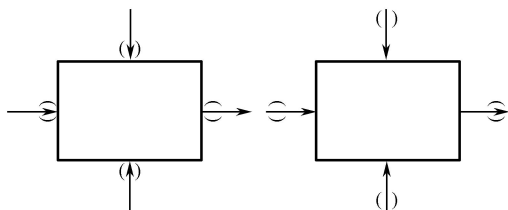


图 5.16 通道箭头的表示

#### 4. ICOM 码

ICOM 是 Input, Control, Output, Mechanism 的缩写。ICOM 码是对图形中的每个活动的箭头规定的编号方式, 用专门的符号说明父子图中的箭头关系, 并把子图中每个边界箭头的开端分别用字母 I, C, O, M 来标明是父盒子的输入、控制、输出及机制, 再用一数字表示父盒子上箭头的相对位置, 编号次序是从上到下、自左到右, 如图 5.17 所示。

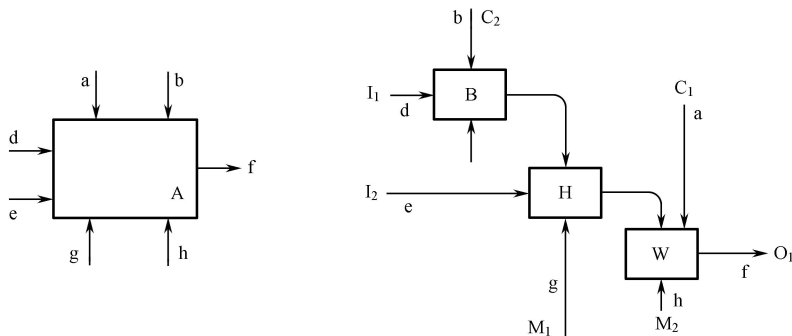


图 5.17 ICOM 码示例

#### 5. 节点号

经过组成分解的一个 IDEF0 模型是一组有一定层次的图形。可以用节点的编号来标志活动所在层次中的位置, 节点的编号是逐层推导出来的。

活动图的所有节点号都用字母 A 开头。最顶层图形为  $A_0$ , 在  $A_0$  以上只用一个盒子来代表系统内外关系, 编号为  $A-0$ 。每个节点号是把父图的编号与父模块在父图中的编号组合起来的。每增加一层, 节点的编号增加一位, 形成如图 5.18 所示的节点树。

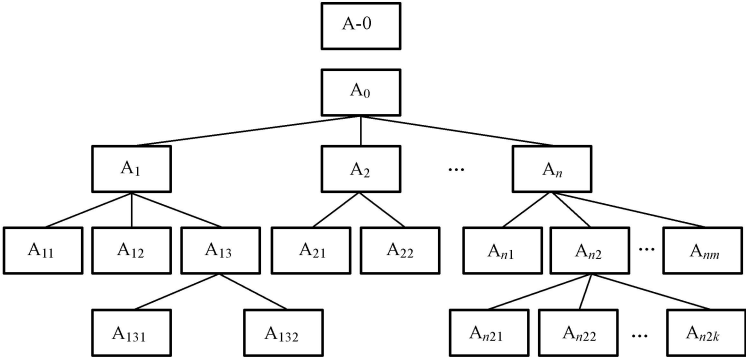


图 5.18 节点树

6. 模型名

每个模型有一个名字。通常用名字代表主体，用子名字表示不同的模型。名字与子名字间用斜杠 (/) 隔开。

7. 如何读图

可以用以下步骤阅读图形。

- (1) 查看当前图中的各盒子，得到描写事物的一个概貌。
- (2) 参看父图，注意有关箭头。试着识别一个最重要的输入、控制及最重要的输出。
- (3) 回到当前图，试着确定是否有一条主路径连接最重要的输入或控制，达到最重要的输出。
- (4) 把主路径作为线索从左上角到右下角遍历这个图形。注意与盒子有关的其他箭头。确定是否存在第二条路径。检查图形所表明的全部内容。
- (5) 最后阅读补充文本，使理解更加完整。

5.4.3 IDEF0 建模过程

1. 选择范围、观点及目的

在开始建立模型之前，首先应确定建模对象的立足点，主要包括确定建模的范围、观点及目的。

范围把模型的主题作为更大系统的一部分来看待，描述了外部接口，区分了与环

境之间的界线，确定了模型中需要重点讨论的问题与不应在模型中讨论的问题。

观点是从哪个角度去观察建模对象，以及在限定范围内涉及的各个不同的组成部分。

目的确定了模型的意图或明确其交流的目标，说明了建模的原因，如功能说明、实现设计及操作等。这三个概念指导并约束了整个建模过程。虽然在建模过程中这些内容也可以有所变化，但必须是自始至终地保持一致、清晰，而不被曲解。

## 2. 建立内外关系图——A-0 图

建模的第一步通常是建立内外关系图——A-0 图。画一个单个的盒子，里面放上活动的名字，名字要概括所描述系统的全部内容。再用进入及离开盒子的箭头表示系统与环境的数据接口。这个图形确定了整个模型的内外关系，确定了系统的边界，构成了进一步分解的基础。

## 3. 画出顶层图

把 A-0 图分解为 3~6 个主要部分，得到  $A_0$  图， $A_0$  图表示了与 A-0 图同样的信息范围。

$A_0$  图是模型真正的顶层图，它是第一个也是最重要的一个，从结构上反映了模型的观点。 $A_0$  图的结构，清楚地表示了 A-0 盒子的名字所要说明的含义。比  $A_0$  图更低级的图形，用以说明  $A_0$  中各个盒子所要说明的内容。

## 4. 建立一系列图形

为了形成图形结构，把  $A_0$  图中每个盒子处理得跟 A-0 盒子一样，即把它们分解成几个主要部分来形成一张新图。

分解的次序可采用以下原则：

- (1) 保持在同一水平上进行分解——均匀的模式深度；
- (2) 按困难程度进行选择，即从最困难的部分开始，选择某一盒子进行分解。

## 5. 写文字说明

最后每张图将附有 1 页（特殊情况可增加）的叙述性文字说明。文字说明分成两列，左边一列为“说明（Text）”，右边一列为“词汇表（Glossary）”。

## 6. 绘制 IDEF0 图示例

这里，以银行支付购书款功能来加以说明。该功能根据购书人提出的购书请求，

银行根据账户使用人的账户信息、银行规定和预算情况，填写账簿和进行付款。其最上层 IDEF0 图如图 5.19 所示。其中，活动为处理购书账户支付，输入购书请求，在银行规定和预算的控制下，以个人账户信息为依据，产生修改以后的账簿和进行付款。

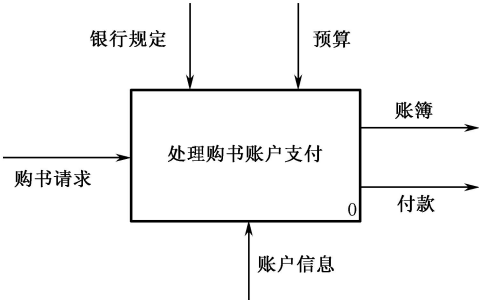


图 5.19 购买图书顶层图

在图 5.19 的基础上，根据实际业务规定，可以将该图分解成为三个子活动，即处理请求、处理发票和填写账簿。经过细化后得到图 5.20。

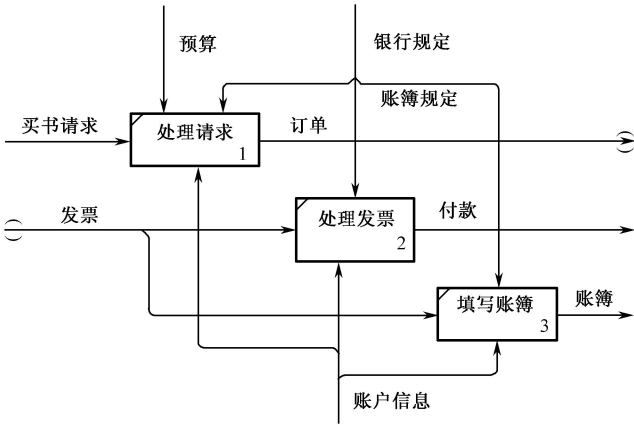


图 5.20 购买图书细化图

其中，账户信息作为机制同时作用在三个活动上，注意输入项发票和输出项订单是通道箭头，说明这两项与子图无关，在其他子图中具体定义。

本例中分解后的功能已经明确，不需要进一步分解。



#### 5.4.4 IDEF0 分析方法与数据流分析方法的比较

IDEF0 分析方法和数据流分析方法两者的异同点如下所述。

##### 1. 相同点

(1) 功能建模方法，用于描述系统的功能。

(2) 结构化方法，采用自顶向下、逐层分解的原则。顶层图是对系统的最抽象的描述，底层图是对系统功能的详细描述。

(3) 在功能分解过程中，保持父图和子图的平衡、一致。

##### 2. 不同点

(1) 数据流分析方法描述系统的数据及其处理和流动的情况，IDEF0 不局限于此。事实上，IDEF0 研制的最初目的在于表达 CIM (Computer Integrated Manufacture) 中的活动。

(2) 模型中表示的图形符号不同。IDEF0 中只有活动和箭头（输入、输出、控制、机制）符号；数据流分析方法中有处理、外部项、数据存储和数据流。

(3) IDEF0 中的活动符号和数据流分析方法中的处理符号表示含义和概念的外延不完全相同。IDEF0 中的活动盒子中，输入可以是数据、事物等，而在数据流分析方法中仅允许是数据；同样 IDEF0 活动的输出较数据流分析方法处理输出的外延要广。

(4) IDEF0 活动受到控制和机制箭头的约束和说明，而数据流分析方法中则没有。因此，IDEF0 描述的功能较数据流分析方法描述的功能更加详细和全面。

(5) 数据流分析方法中的数据存储是 IDEF0 中所没有的。

(6) 作为数据流分析方法的补充说明——数据字典较 IDEF0 的补充说明详细得多。

综上所述，IDEF0 建模分析方法在对系统功能的描述方面较数据流分析方法更加全面、翔实，而数据流分析方法在对系统的数据描述方面更加详尽。

### 5.5 逻辑分析工具

结构化系统分析的基本思想是将一个大型复杂的系统最终分解成许多个足够简单的基本处理过程。前面已经讨论了上述系统分解的 IDEF0 图、数据流图和描述系统内数据的数据字典等，本节重点讨论对处理过程的功能分析。

功能分析的任务是把数据流图中各个处理过程的功能加以详尽的说明，并精确地描述用户要求一个处理过程“做什么”，这包括处理过程的激发条件、处理逻辑、容错处理等。其中最基本的部分是处理逻辑，即用户对这个处理过程的逻辑要求，以及该过程的输出数据流与输入数据流之间所具有的逻辑关系。

分析阶段的任务是理解和表达用户的要求，构造新系统的逻辑模型，而不是具体考虑系统如何去实现。因此，对一个处理过程应描述的是做什么，而不是用计算机编程语言来描述具体的操作过程，如内存分配、执行控制等，也不涉及具体的物理工具和设备。

处理过程中对数据的所谓处理和加工，一般包括以下三个含义。

- (1) 数学运算：对输入数据进行数学变换，通过数学工具予以表达。
- (2) 数据交换：与数据存储或外部实体进行信息交流。
- (3) 逻辑判断：根据判别各种条件的结果，执行不同的操作或采取不同的行动。

数学运算与数据交换就性质而言，都是可以用一种精确的语言予以表述的，但逻辑判断的表达常常可能涉及一些非精确的、意义不明确的描述，反映一种决策的选择。为此，结构化系统分析方法采用了若干种决策分析工具，来对逻辑判断作出描述。这些工具并非严格的形式语言，但能够比较明确地把用户的要求表达出来，易于为用户所理解。

进行决策是业务的一个组成部分。其实，管理本身实质上就是进行决策。对所有的决策来讲，共同的概念是当分析决策时，应从识别条件和行动开始。在决策分析中，必须同时识别在某一情况下发生的有关条件和容许条件，并在研究时注意将有关的这些条件都包括进去。当所有的可能条件都已经知道后，分析人员就须确定当一定条件发生时应该做什么，即行动（Action）。

下面将通过一个例子来帮助对该问题的说明。某企业根据推销人员所推销出去产品的价值来确定给予相应的奖励，具体方法是：按照推销出去的产品金额减去事先确定的推销指标为条件建立起三种不同的奖励政策。若超额部分大于或等于 100 000 万元，则奖励超额部分的 1%；若为 50 000 元到 99 999 元，则奖励 0.8%；小于 50 000 元的奖励 0.5%；若推销人员没有完成推销指标，则不予奖励，见表 5.3。

表 5.3 某企业推销奖励政策

条 件	行 动
超额部分：超过 100 000 元	奖超额部分的 1%
50 000~99 999 元	奖超额部分的 0.8%
50 000 元以内	奖超额部分的 0.5%
没完成指标	不予奖励

在许多决策中，分析人员必须考虑条件和行动的组合。为了理解这些组合，可以使用下面介绍的决策树、决策表和结构式语言等逻辑分析工具。

### 5.5.1 决策树

决策树 (Decision Tree) 是一种图形，它能顺序地表示出条件和行动，因而能显示出应首先考虑哪些条件，其次考虑哪些条件等。它也表示出各条件和所允许的行动的关系。由于图形很像树枝，因此称其为决策树。

决策树图形的左边是树根，它是决策序列的起点。紧跟着的是各个分支，它们依赖于存在的条件和所作的决策，如图 5.21 所示。树中非叶节点代表条件，它指出必须在能够选择下一条路线之前作出决定，查看条件是否满足，并依据条件作出决策。树的叶节点表明要采取的行动，这种行动是依赖于其左边的条件序列的。从树根开始，自左至右地沿着某一个分支，能够作出一系列的决策。

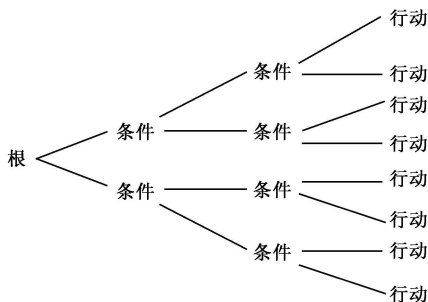


图 5.21 决策树的一般形式

以前述推销奖励政策为例。当企业确定奖励时，还要规定推销员推销出去的产品必须在本年度收到对方的购货支票，即对方未付款的不列入奖励范围。那么，这个问题的决策顺序如图 5.22 所示。



图 5.22 用来确定奖励政策的决策树

对决策分析来说，决策树并不经常是最好的工具。当决策问题太复杂时，会存在许多步骤和组合条件的序列，结果系统的规模变得难以控制。分支的数目太大和通过的路径太多，对分析不但没有帮助而且会使分析人员束手无策。在发生这些问题的场合，分析人员应避免用决策树，而考虑改用决策表。

5.5.2 决策表

决策表（Decision Table）是显示条件和行动的一个行列矩阵，而不是树。决策表中还包括决策规则，它说明当某些条件成立时，紧跟的是什么行动。

决策表由四部分组成：条件语句、条件项、行动语句和行动项。条件语句部分用于识别有关的条件。条件项指出，结果需要什么值加到各个条件上。行动语句列出了某些条件出现时应采取的所有步骤。行动项表示，当选择的条件或条件的组合成立时，应采用哪些具体行动。

在决策表的右边各列，条件与行动相连组成决策规则（Decision Rules）。决策规则说明了对所采取的特定行动必须满足的条件。决策规则体现所有必须成立的条件，而不是每次只体现一个条件。

表 5.4 所示是图 5.22 的决策表。在该表的条件项部分，用 Y（是）或 N（非）代表每一个条件是存在还是不存在。行动项部分用 X 表示选择的行动。

表 5.4 用来确定奖励政策的决策表

	条 件	决策规则
条件 语句	本年度付款	YYYY NNNN
	超过 100 000 元	YNNN YNNN
	50 000~99 999 元	NYNN NYNN
	50 000 元以下	NNYN NNYN
	未完成指标	NNNY NNNY
行动 语句	奖励 1%	X
	奖励 0.8%	X
	奖励 0.5%	X
	不予奖励	X XXXX

每个决策表构成以后，系统分析人员要检验它的正确性与完整性，以保证决策表能包括所有条件及决策规则。另外，分析人员还应消除表中可能有的冗余项和互相矛盾的决策规则。

例如，在表 5.4 中，当付款是在本年度以外时，超额再多也是不予奖励的。由于这个事实，决策规则就冗余了，可以组合成一个规则，因此，可将表 5.4 所示的决策表改进为表 5.5。

表 5.5 改进的用来确定奖励政策的决策表

	条 件	决策规则
条件 语句	本年度付款	YYYY N
	超过 100 000 元	YNNN —
	50 000~99 999 元	NYYN —
	50 000 元以下	NNYN —
	未完成指标	NNNY —
行动 语句	奖励 1%	X
	奖励 0.8%	X
	奖励 0.5%	X
	不予奖励	X X

### 5.5.3 结构式语言

结构式语言 (Structured Language) 是介于自然语言和形式语言之间的一种半形式化语言, 实质上它是自然语言的一个受某些限制的子集。选择结构式语言作为处理过程逻辑要求的描述语言, 是因为它具有自然语言简单易懂的优点, 又可避免自然语言的一些缺点, 与自然语言的不同之处在于它只使用了极其有限的词汇和语句, 与形式语言的不同之处是它没有严格的语法规定。

结构式语言只使用以下几类语句:

- (1) 简单的祈使语句;
- (2) 判断语句;
- (3) 循环语句;
- (4) 上述三种语句的复合语句。

#### 1. 祈使语句

祈使语句明确地指出做什么事情, 它至少包括一个动词说明要执行的功能及一个名词表示动作的对象。

例如, 人们到书店去买书, 常常会按下述过程处理:

- (1) 选择一本满意的书籍;
- (2) 携带该书到付款台;
- (3) 付款;
- (4) 盖付款标记;
- (5) 离开书店。

上例即是用结构式语言描述的买书过程, 其中的每一条语句都是祈使语句, 并按

顺序显示出 5 个步骤。步骤中没有包括任何一个决策或条件，仅按次序列出。每一个步骤都有特定的次序，乱了次序，买书过程就不成立了。所以，对处理过程的描述必须指出行动的正确次序。

## 2. 判断语句

判断语句就是为描述决策结构而设计的语句，其一般形式如下：

```

如果 条件
    则 行动 A （条件成立）
    否则 行动 B （条件不成立）
  
```

其中，行动 A 与行动 B 可以是一组祈使语句，或是循环语句，甚至是另外一个判断语句。

例如，判定学生考试成绩时，如果得分为 90~100 分，则评为 A；80~89 分为 B；70~79 分为 C；60~69 分为 D；59 分以下为 E。其结构式语言如下：

```

如果 成绩为      分
    则评为
    否则 如果      分
        则评为
        否则 如果      分
            则评为
            否则 如果      分
                则评为
                否则 如果 分以下
                    则评为
  
```

还有一种多种选择结构，它使用选择/情况词组。其一般形式如下：

```

选择 合适的情况
    情况 1：行动 A
    情况 2：行动 B
    ...
    情况 n：行动 N
    否则：行动 W
  
```

## 3. 循环语句

循环语句是在某一条件存在时，重复执行相同的行动，直至该条件不成立为止。

其一般形式为：

当 条件 做  
行动 A

例如，教师给学生判考试卷及评定成绩时，通常连续、重复地对每一张试卷判分评定，其结构式语言描述如下：

当 还有未判试卷 做  
判定试卷得分  
评定成绩等级

其中“评定成绩等级”就是前面用判断语句描述过的行动。

#### 5.5.4 三种逻辑分析工具的比较

决策树、决策表和结构式语言这三种逻辑分析工具各有优缺点。在表达一个处理过程的时候，系统分析人员应根据不同的情况，选择合适的逻辑分析工具。

表 5.6 根据可理解性、可验证性、直观性、易设计性、可修改性及自动生成等指标，对这三种逻辑分析工具进行了比较。

表 5.6 三种逻辑分析工具的比较

工具	可理解性	可验证性	直观性	易设计性	可修改性	自动生成
决策树	好	较差	好	较差	较好	较差
决策表	较差	好	较差	好	较差	好
结构式语言	较好	较好	较好	好	好	好

根据上述分析，可以得出如下结论：

- (1) 对一个不太复杂的逻辑判断，使用决策树较好；
- (2) 对一个十分复杂的逻辑判断，使用决策表较好；
- (3) 如果一个处理过程中既包含顺序结构，又有判断和循环逻辑，则使用结构式语言较好。

## 5.6 系统设计概述

经过系统分析，对原系统的业务处理过程、数据流程、数据特征、处理功能及存在的问题等有了较深入的了解，从而提出了系统的逻辑模型。从建立模型的角度来看，系统分析提供的逻辑模型只解决了系统要“做什么”的问题，而究竟如何做才能达到系统的目标，这在系统分析阶段并没有得到解决。

系统设计就是为实现系统分析提出的系统逻辑模型所作的各种技术考虑和设计。系统设计又称为系统的物理设计，即根据新系统的逻辑模型建立系统的物理模型，也即根据新系统逻辑功能的要求，考虑系统的规模和复杂程度等实际条件，进行若干具体设计，确定系统的实施方案，解决系统“怎么做”的问题。因此，系统设计是开发信息系统的最重要的环节之一。这一阶段工作的优劣，直接影响信息系统的性能、功能、效率和效益。

### 5.6.1 系统设计的任务

系统设计工作是实施系统的依据，它在充分反映系统分析阶段的成果（系统的逻辑模型）的基础上，从技术上对新系统进行反复论证，把整个系统规定在更现实、更合理的范围之内，使今后的工作更加切实有效，避免盲目性。系统设计阶段的主要任务包括：

- （1）计算机系统及其他硬件设备的选择；
- （2）系统的分解与组织；
- （3）原始数据的组织和输入；
- （4）输出信息的方式和管理；
- （5）文件与数据库的组织和管理；
- （6）编码的设计与确定；
- （7）通信网络的设计；
- （8）系统的安全保密性设计；
- （9）系统实施计划；
- （10）其他。

反映系统设计成果的是《系统设计说明书》，它是系统建设的必备文件。系统设计说明书从系统设计的诸主要方面说明系统设计的指导思想及采用的技术方法，既是上级管理部门审查和协调的依据，也是系统实施的重要依据。

### 5.6.2 系统设计的目标

要建立一个新的信息系统，我们总是期望它比原有的系统在以下诸方面能有较大的改进。

- （1）更快、更准、更多地提供资料；
- （2）更多、更细的处理功能；
- （3）更有效、更科学的管理方法。



概括地说,系统设计的目标是:在保证实现逻辑模型的基础上,尽可能地提高系统的各项指标,即系统的运行效率、可靠性、可修改性、灵活性、通用性和实用性。

为了实现上述要求,需要设计人员从若干种设计方案中选取最佳的设计方案,选择的依据是根据设计目标来评价设计方案。我们重点介绍系统的运行效率、可靠性和可修改性。实际上,这三项是系统设计中的最主要的设计目标。

### 1. 系统的运行效率

系统的运行效率包括三个方面的内容。

- (1) 处理能力:指在单位时间内能够处理的事务个数。
- (2) 处理速度:指处理单个事务的平均时间。
- (3) 响应时间:指从发出处理要求到给出回答所用的时间。

不同处理方法的系统,其运行效率有不同的含义。例如,联机实时处理系统的运行效率主要用响应时间来表示;批处理系统的运行效率常用处理速度来表示;而对于一个数据库系统,通常希望它有较高的处理能力和响应时间等。

影响系统运行效率的因素一般取决于下述两个方面。

#### (1) 系统中的硬件及其组织结构

系统中的硬件及其组织结构,对系统的运行效率有直接的影响。一方面,硬件可利用的资源越多,系统的运行效率越高。这些资源包括中央处理机(CPU)、内存存储器、外存储器和输入/输出设备等。由于任何一个系统可用的资源总是有限的,因此重点是设法提高这些资源的使用效率。另一方面,硬件设备的处理时间对系统的运行效率有很大的影响,这里指的是处理机的运行时间、外部设备的运行时间和通信线路时间等。

#### (2) 计算机处理过程的设计质量

计算机处理过程的设计质量,标志着系统设计方案的优劣,也直接影响着系统的运行效率。我们从以下几个方面简单说明这一问题。

① 中间文件的数量。中间文件只是用于暂时存储有关数据,并不具有保存价值。中间文件通常由某一模块产生,交由下一模块使用。使用完后,往往就可抛弃它。建立过多的不必要的中间文件和反复读取这些文件必然造成系统运行效率的降低。

② 文件的存取方法。文件的存取方法是由文件记录的组织方式及硬件设备的特点所决定的。选择合适的文件组织方式及存取方法能在很大程度上提高系统的运行效率。有关这方面的内容,应在数据库设计中详述。

③ 子程序的安排及软件编制质量。系统中常常有很多子程序,它们各自担负着某一特定功能的实现。对于经常要用的子程序,应安排它们常驻内存,否则反复读取外存需要花费很多系统时间。另外,软件编制质量的好坏,也会影响程序的执行时间。

## 2. 系统的可靠性

系统的可靠性是指在系统运行过程中，抵抗异常情况（人为的和机器的故障）的干扰、保证系统正常工作的能力。它包括系统硬件和软件的可靠性。一个可靠的系统，在合理状态下使用时，是不会引起危险和严重失灵的。系统的可靠性包括：检、纠错的能力，对错误的容忍能力，排除错误的能力等。衡量系统可靠性的一个重要指标是系统的平均故障间隔时间（MTBF）和平均维护时间（MTTR）。前者指平均的系统前后两次发生故障的间隔时间，反映了系统安全运行的时间；后者指发生故障后平均每次修复所用的时间，反映了系统可维护性的好坏。系统的平均故障间隔时间越长，系统的可靠性就越高；系统的平均维护时间越短，则系统的可维护性越高。

系统的有效性可用下式表示：

$$\text{系统的有效性} = \text{MTBF} / (\text{MTBF} + \text{MTTR})$$

因此，平均故障间隔时间越长而平均维护时间越短，系统的有效性越高。

提高系统可靠性的途径主要有：

- （1）设计中尽可能地避免出错；
- （2）对可能出现的错误，系统要有完善的检、纠错功能和对安全性的考虑，如对输入数据进行检验、建立运行日志和监督跟踪、规定文件存取权限及适时备份等；
- （3）对可能的错误进行出错冗余设计，如硬件选择及备选、软件的容错能力等。

## 3. 系统的可修改性

系统的可修改性是指系统容易修改的程度。它并没有一个定量的标准，而是通过比较得出的结果。但是，应用合理的系统设计原则，能够对系统的可修改性产生积极的影响。例如，模块的独立性、耦合、聚合等原则在设计中的应用，都能使系统各部分的独立性增强，容易进行改动，从而满足对系统目标的变化要求。

系统的可修改性之所以重要，是因为一个系统从设计到建成运行，总是处于不断的变化之中，这就必然引起修改和维护。这些变化包括：

- （1）系统环境的变化，引起系统目标的改变或部分改变，这时就要对系统进行扩充和改进；
- （2）现代信息技术（如计算机技术、管理决策方法等）的发展，必然对系统产生影响，如引进新设备、使用新方法等引起的系统的改变；
- （3）系统本身总是处于不断的完善之中，不适应或错误在所难免，这也要求对系统进行修改和维护。

提高系统的可修改性，除了设计人员的知识与经验外，还必须在系统的分析与设

计过程中,始终采用结构化、模块化的方法和原则。这些方法的采用,将有利于提高系统的可修改性。

上述系统设计的目标,在一定程度上既是相辅相成的,又是相互矛盾的,合理地运用和协调好这几个方面,才能使系统具有较高的生命力。

### 5.6.3 计算机处理与手工处理

在系统设计过程中,始终要明确应用计算机处理和手工处理的界限。信息系统是人机系统。系统目标的实现取决于这两者的结合,系统设计中要避免这样两种倾向:

(1) 一味地追求计算机处理,将许多只能由人完成的工作交由计算机处理,从而造成设计的复杂和不够科学;

(2) 把本该由计算机完成的工作交由人去处理,从而使新系统的功能、性能及用户的目标得不到体现。

计算机和其他工具一样,只能按照人设计或设想的方式去工作,按照人给定设计好的功能去完成任务。然而,人们在处理一切事务中,很难事先规定出严格的逻辑次序来对付一切问题,总会发生“预料之外”的事情,遇到这种情况,就不得不依靠人的经验和知识加以判断,选择适当的处理方法。这些经验如何用计算机表达,如何抽取出来对付所发生的问题,目前还不能完全解决。因而,在计算机应用中,总是需要人的干预。这种干预大多在系统控制与决策部分。

在实现系统逻辑功能的物理设计中,系统设计人员要划分出哪些功能由计算机完成,哪些功能用人工处理,这就需要了解两种处理过程的不同特点,如下所述。

(1) 处理速度不同:对一般的事务处理,计算机处理要比人工处理快得多。

(2) 概念的精确程度不同:计算机处理问题时,必须有明确的概念,模糊的处理还很难做到。随着决策科学和模式识别等技术的不断发展,计算机将能在许多领域进行模糊处理。

(3) 对处理的信息的结构化程度的要求不同:结构化程度(指信息格式固定、长度固定、各种信息之间的从属关系固定等)越高,计算机越易处理。对结构化程度低的信息,计算机就很难判断,但人工处理很容易。

(4) 工作效率的稳定性不同:用计算机处理数据,速度一般是固定的,但人工处理却有很大的差异。计算机所处理的问题,其计算的结果比人工处理的准确、稳定,人工因受能力、情绪等限制而易出错。

(5) 意外情形的处理:遇到这种情况,计算机往往不如人灵活,因为计算机的判断、决策能力不强。

(6) 费用的考虑：使系统的成本合适，并使人和计算机都能充分发挥作用。

综上所述，划分计算机处理与人工处理的基本原则是：

(1) 复杂的科学计算、大量重复的数学运算、统计、汇总、报表、数据库检索、分类、文字处理、图形图像基本处理、有关数据的采集、通信等应由计算机完成。

(2) 对于传统的人工判定，目前没有成熟的技术可以应用，或代价太高，故仍用人工处理。

(3) 决策性问题中，计算机尽可能提供决策依据，由人进行最后决策。

(4) 设计人机接口，考虑时间的匹配、代码的统一、格式的协调等。

## 5.7 结构化设计原理

与结构化分析的方式相似，在系统设计中采用自顶向下逐层分解、逐步求精的方式进行即结构化设计。本节介绍结构化设计的方法和原理。

### 5.7.1 结构化设计方法

#### 1. 结构化设计思想

早期的计算机应用，由于受到软硬件的限制，因此只是编制了一些简单程序，其设计工具是程序流程图。但是计算机技术的发展及应用领域的不断扩大与复杂，使得程序流程图已不能满足设计的需要。

20 世纪 70 年代以来，出现了多种设计方法，其中有代表性的是结构化设计方法、Jackson 方法、Warnier 方法和 Parnas 方法等。尤其结构化设计方法是在结构化程序设计思想的基础上发展起来的用于复杂系统的设计技术，其最重要的思想就是对问题要有一个总的看法，由概括到具体、逐层分解问题。结构化设计方法强调把一个系统设计成具有层次式的模块化结构，并且用一组标准的准则和工具帮助系统设计人员确定组成系统的模块及相互关系。结构化设计采用先全局后局部、先总体后细节、先抽象后具体等过程开发系统，从而使系统结构清晰，可读性、可修改性、可维护性等指标优异。

#### 2. 结构化设计方法的特点

概括地说，结构化设计方法有以下几个特点。

(1) 对一个复杂的系统，应用自顶向下、逐步求精的方法予以分解和化简。

(2) 强调采用模块化的设计方法，并有一组基本设计策略。

- (3) 采用结构图作为模块设计的工具。
- (4) 有一组评价设计方案质量的标准及优化技术。

### 3. 结构化设计方法的主要内容

结构设计方法的主要内容如下所述。

(1) 合理地进行模块分解和定义，使一个复杂系统的设计转化为若干个基本模块的设计。结构化设计的分解原则是：

- ① 把密切相关的子问题划归为系统的相同部分；
- ② 把不相关的问题划归为系统的不同部分。

(2) 有效地将模块组织成一个整体，从而体现系统的设计功能。

结构化系统设计与结构化的系统分析有着密不可分的联系，它是以系统的逻辑模型和数据流图为基础，借助于一套标准的设计方法和图表工具，通过自顶向下或自底向上的方法，逐层把系统划分为多个大小适当、功能明确、具有一定独立性的模块的。因此，模块的组织是其主要内容。

## 5.7.2 结构化设计原理

### 1. 模块化原理

依据系统是由元素和结构组成的，且它们又可分层、分类的思想，一个信息系统可被逐层划分为大小适当、功能明确独立且容易实现的模块，并由它们的协调和组合去共同达到系统的设计目标。这就是结构化设计方法的模块化原理。

模块化（Modularization）原理的基本思想是可分性，即最大限度地降低系统设计的复杂性，把问题层层分解，直到容易解决为止。这样就降低了人为划分的主观影响，增强了问题的客观性。

### 2. 信息隐蔽原理

模块化原理告诉我们，任何复杂的信息系统都是可分的。但到底怎样分才最有效呢？信息隐蔽原理是指导人们按信息相关度划分模块的一个原理。

信息隐蔽（Information Hiding）原理的基本思想是：在一定规模和条件的限制下，把那些与模块功能相关度最大的信息（如过程与数据）分在一个模块内，而把最少的完成功能所需的交互信息划分在该模块外（接口）。换言之，模块的划分应该使得包含在模块内的信息对于无须这些信息的模块是不可访问的。

模块是构成系统的元素，元素要完成自己的功能必须有它的实体，但同时又必须

保持对其他元素的开放，这样才能使元素既具有相对独立的功能又保持与系统和环境的交流，从而像有机体的细胞一样共同承担系统的总体功能。信息隐蔽原理只是在继承这一思想的基础上，进一步要求使必要的外部联系最少、使内部的联系充分得多而已。

事实上，信息隐蔽原理不仅适用于软件系统的模块划分，也适用于硬件系统的模块划分及其他系统的子系统划分，它是把系统划分成子系统的普遍性原理。遵循这一原理的好处是显而易见的：一方面最大限度地体现了模块在概念和功能上的独立性、相对完整性和封闭性，另一方面又给系统结构和模块的实现、维护带来了方便。隐蔽的直接好处是，模块内部的错误很少可能传播到系统中的其他模块。

### 3. 时空等价原理

时空等价（Space-time Equivalence）原理是在更高层次上（包括系统层）指导人们按时空关系划分子系统或模块的一般性原理。

信息系统的直接资源主要是计算机软件、硬件资源。我们知道，在逻辑上，软、硬件资源是等价的。相对而言，软件更耗费时间而不耗费空间，硬件则更耗费空间而不耗费时间。换言之，对完成同样的功能，软件比起硬件来耗费的空间可以忽略，硬件比起软件来耗费的时间可以忽略。

时空等价原理有三方面的含义：一是从理论上讲，对于一个特定的系统功能或模块功能，既可由硬件模块去完成，又可由软件模块去完成，此即功能等价；二是硬件有耗费空间但速度快的属性，而软件有耗费时间但不耗费空间的属性；三是由于任何一个信息系统总是希望占用或耗费最少的空间与时间，因此为了达到这个目标，必须对系统的资源合理分配，使同一功能由耗费时空最少的模块或子系统去承担。

运用时空等价原理划分硬、软件子系统或模块时，必须按系统与环境的约束条件在系统硬、软件的时空属性中进行综合折中。此时，就会应用时空的另一个重要原理：时空权衡原理。

时空权衡（Space-time Trade-off）原理是指：牺牲空间或者其他替代资源，通常都可以减少时间代价；反之亦然。

## 5.8 模块化设计

由上述已知，结构化设计的主要思想和原理就是模块化。下面介绍模块的概念、模块的耦合和模块的聚合。

### 5.8.1 模块

前面我们已经使用了模块的概念,实际上,模块化的概念在计算机科学中已经使用了几十年。在传统的程序设计中,能够执行某项功能的若干条程序语句,都可看作一个模块。但在系统设计中,这种定义就不够准确、完整。下面给出模块的定义。

所谓模块 (Module),是指这样的一组程序语句 (或描述),它包括输入与输出、逻辑功能描述、内部信息及其运行环境。

(1) 输入与输出:模块的输入来源和输出去向在正常的情况下都是同一个调用者,即模块。从调用者处获得输入信息,经过模块本身的处理后,再把输出返送给调用者。

(2) 逻辑功能:模块的逻辑功能描述了该模块能够做什么样的事情,具备什么样的功能,即对于输入信息能够加工成什么样的输出信息。

(3) 内部信息:模块的内部信息是指模块执行的指令和在模块运行时所需要的属于该模块自己的数据。

(4) 运行环境:模块的运行环境说明了模块的调用与被调用的关系。

在系统设计中,我们只关心模块的外部信息,即研究模块能完成什么样的功能,具体的实现将在系统实施阶段完成。除了上述四个特性外,一个模块通常还有其他一些属性,如模块的名称、编号等。

#### 1. 模块化

所谓模块化,简单地说就是把系统划分为若干个模块,每个模块完成一个特定的功能,然后将这些模块汇集起来组成一个整体 (即系统),以完成指定的功能,满足问题的要求。

引入模块化概念,就是为了使问题的解决变得容易。

设对于两个问题  $P_1$ 、 $P_2$ ,  $C(P_1)$ 、 $C(P_2)$  表示问题的复杂程度,而  $E(P_1)$ 、 $E(P_2)$  表示解决问题所需的工作量。于是,如果

$$C(P_1) > C(P_2)$$

则

$$E(P_1) > E(P_2)$$

一个有趣的规律是

$$C(P_1 + P_2) > C(P_1) + C(P_2)$$

即如果一个问题由两个问题组合而成,那么它的复杂程度大于分别考虑每个问题时复杂程度之和。于是

$$E(P_1 + P_2) > E(P_1) + E(P_2)$$



推而广之, 当一个系统有  $n$  个独立元素时, 一般总有

$$C(P_1 + P_2 + \cdots + P_n) > C(P_1) + C(P_2) + \cdots + C(P_n)$$

及

$$E(P_1 + P_2 + \cdots + P_n) > E(P_1) + E(P_2) + \cdots + E(P_n)$$

这就是模块化的依据。上述论述说明了把复杂的问题分解为若干容易解决的子问题后, 原来的问题也就容易解决了。

尽管上述论证还很不精确, 但可以说明模块化设计的重要性, 那么能否认为, 如果我们无限地分割系统, 最终将导致最基本模块的设计非常容易, 因而使得设计系统的工作量非常小呢? 事实上, 这个结论是错误的。如图 5.23 所示, 当模块数目增加时, 每个模块的规模将减小, 开发单个模块所需的成本 (工作量) 确实减少了, 但随着模块数目的增加, 设计模块间接口的工作量也将增加。因此, 每个系统都存在一个最适当的模块数目, 使得系统开发成本最低。

采用模块化设计原理可以使整个系统设计简易, 结构清晰, 可读性、可维护性增强, 提高系统的可靠性, 同时也有利于信息系统开发工作的组织和管理。

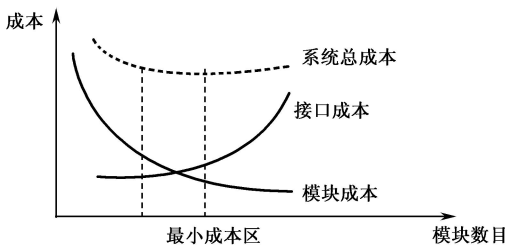


图 5.23 模块化和系统成本

## 2. 模块独立性

系统设计强调把一个系统设计成具有层次式的模块化结构, 前面讨论了诸如系统分解、模块化等问题。人们希望获得这样的一种系统结构:

- (1) 每个模块完成一个相对独立的特定功能;
- (2) 模块之间的接口简单。

系统设计的目标是尽量寻求一个合理的模块分解数目。按照前面的考虑, 其中 (1) 保证了按模块分解后的设计工作量将小于按整个系统设计的工作量, (2) 则试图使设计中的接口成本最小。

下面我们引入模块独立的概念。具有独立功能而且与其他模块之间没有过多相互



作用的模块，我们称之为独立的模块。所谓两个模块彼此完全独立，是指其中任意一个模块在运行时，与另一个模块的存在与否完全无关。当然，独立只是一个抽象的、相对的概念。既然各个模块隶属于一个系统，那么它们之间就必然存在或多或少的某种联结，从而使得它们组成一个有机的整体。

一般地说，模块之间的联系越多或越复杂，它们之间的相互依赖程度就越高，因而每一个模块的独立性就越低，这显然是不可取的。那么，为什么模块的独立性很重要呢？

第一，具有独立性的系统比较容易开发，这是由于能够分割功能而且接口可以简化的原因。系统开发常常是由若干人的分工合作完成的，这时具有模块独立性的优点就尤为重要。

第二，模块的独立性越好，模块的相互影响就越少。当系统中某一模块出错时，产生连锁反应的概率就越低，从而提高了系统的可靠性。

第三，独立的模块比较容易测试和维护。这是因为错误的传播范围小，比较容易定位，而且对一个模块进行修改或维护时，不必担心其他模块的内部是否会受到影响。

总之，模块的独立性是设计一个“好”的系统的关键。

模块的独立程度可由两个定性标准度量，这两个定性标准分别称为耦合和聚合。耦合度量不同模块彼此间互相依赖（联结）的紧密程度，聚合则衡量一个模块内部各个元素彼此结合的紧密程度。

(1) 耦合 (Coupling)。耦合是对一个系统内不同模块之间互连程度的度量。耦合强弱取决于模块间的联结形式及接口的复杂程度。模块间接口的复杂性越高，说明联结的程度越高。

模块之间的耦合程度直接影响系统的可读性、可维护性及可靠性。在系统设计中，我们应该尽可能追求松散耦合的系统。这是因为，在这样的系统中可以研究、测试、维护任何一个模块，而不需要对其他模块有很多了解。同时，由于模块联系简单，故错误传播的可能性就较小。

(2) 聚合 (Cohesion)。聚合标志着一个模块内部各个元素彼此结合的紧密程度，主要表现在模块内部各个元素为了执行处理功能而组合在一起的程度。

所谓模块内部的元素是指该模块运行程序中的一条指令。我们希望系统中的每个模块具有高度的聚合性，其各个元素都是彼此密切相关的，是为完成一个共同的功能而结合在一起的。模块设计中应尽力避免低聚合，这是基本的原则。

因此，聚合和耦合是相辅相成的两个原则，是进行模块设计的有力工具，模块内的高聚合往往意味模块之间的松耦合。要想提高模块内部的聚合性，往往必须尽量减少模块之间的联系。实践表明，聚合更为重要，设计者应把更多的注意力集中到提高模块内部的聚合性上。

### 3. 模块的特征

模块的主要特征是：抽象，信息隐蔽。

（1）抽象。抽象是人类认识复杂现象过程中使用的最强有力的思维工具之一。抽象就是将一些具有某些相似性质的事物的公共之处概括出来，暂时忽略其不同之处，或者说，抽象是抽出事物的本质特性而暂时不考虑它们的细节。模块正是反映数据与过程的抽象。

在模块化问题求解时，可以提出许多抽象层次。在抽象的最高层次使用问题环境语言、以概括的方式叙述问题的解法；在抽象的较低层次，则可采用过程性方法描述；在抽象的最低层次，用可以直接实现的术语描述问题的解法。

模块化和逐步求精是与抽象紧密相关的概念。信息系统工程的每一步都是对问题求解方法的抽象层次的一次精化。在可行性研究阶段，是把系统作为一个完整的部分考虑的；而在系统分析、系统设计阶段，抽象的程度随之减少；最后，当系统实现后，就达到了抽象的最低层。模块化和逐步求精的方法把面向问题的术语与面向现实的叙述结合起来了，前者是后者的一种抽象。

（2）信息隐蔽。信息隐蔽是模块的另一个重要特征。按照信息隐蔽原理，模块的信息隐蔽是指一个模块内所包含的信息（过程和数据）不允许那些不需要这些信息的外部模块访问。

模块信息隐蔽的结果意味着：系统有效的模块化可以通过定义一组独立的模块来实现，这些独立的模块彼此之间仅仅交换那些为了完成系统功能所必须交换的信息。

## 5.8.2 模块的耦合

影响模块之间联结程度的最主要的因素是模块间的联结形式。通常，两个模块间的联结形式有数据耦合、控制耦合、公共耦合和内容耦合。下面分别讨论这几种耦合形式及它们的联结程度。

### 1. 数据耦合

如果两个模块彼此间通过参数交换信息，而且每一个参数仅仅为数据，那么这种耦合称为数据耦合。

数据耦合是系统中必不可少的连接形式，它是一种最低的耦合，是一种理想的模块连接。实际上，一个系统内可以将所有模块只设计成数据耦合。但尽管数据耦合是一种好的形式，倘若模块之间传递的数据量大，也会产生不利的影响。

## 2. 控制耦合

如果两个模块彼此间传递的信息中有控制信息,那么这种耦合称为控制耦合。

控制耦合与数据耦合很相似,只不过传递参数中一个仅仅为数据,而另一个含有控制信息。严格地讲,上述的定义是一种数据-控制耦合。因为传递的参数中可以有数据,也可以有控制信息。控制耦合往往是可以避免的。可以通过适当的方式,如模块的再分解,而把这种连接转化为数据耦合,但在某些特殊的场合下,控制耦合还有一定的存在必要。

## 3. 公共耦合

如果两个模块彼此之间通过一个公共的数据区域传递信息,则称为公共耦合或公共数据域耦合。公共数据域实际上就是被设计成为多个模块公用数据的区域,如一个共享数据缓冲区、一个数据文件等。图 5.24 所示为公共耦合示例,模块 A、D、F 共用公共数据域内的元素。尽管模块 D 与 F 没有联系,但它们之间存在着公共耦合。

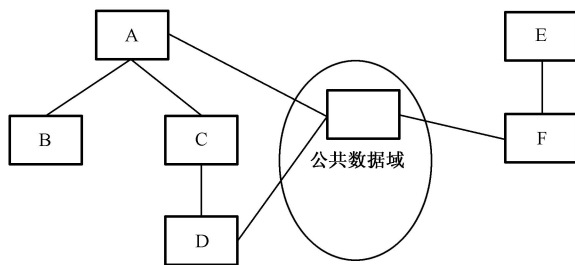


图 5.24 公共耦合示例

公共耦合是一种不好的联结形式,尤其当一个公共数据域被多个模块共同使用时,模块数越多,其耦合的复杂度越大,因为这种情况,给数据的保护、维护等都带来了很大的困难。但是公共耦合可以作为数据耦合的一种补充,如果一个模块与另一个模块需要传递大量的数据时,采用公共耦合就比采用全部传递参数的数据耦合要方便。

## 4. 内容耦合

如果一个模块需要涉及另一个模块的内部信息,则这种耦合称为内容耦合。例如:

- (1) 一个模块直接访问另一个模块的内部数据;
- (2) 一个模块执行另一个模块内部的功能或调用部分程序代码;
- (3) 模块的出、入口不符合单入单出,即有多个不同的入口或出口。

内容耦合的耦合度最高,因此应该坚决避免使用这种耦合。

5. 耦合形式的比较

以上介绍了四种模块耦合，下面对这几种耦合作一番比较，见表 5.7。

表 5.7 四种耦合的比较

耦合形式	可读性	可维护性	扩散错误能力	共用性
数据耦合	好	好	弱	好
控制耦合	一般	不好	一般	不好
公共耦合	最坏	坏	强	最坏
内容耦合	最坏	最坏	最强	最坏

因此，在对一个系统进行模块设计时，我们应当遵循下列原则：

- (1) 模块间尽量使用数据耦合；
- (2) 必要时才采用控制耦合；
- (3) 对公共耦合应限制耦合的模块数；
- (4) 坚决不用内容耦合。

5.8.3 模块的聚合

模块内部的紧凑性主要表现在一个模块内部各组成部分之间的联系上，共有七种不同类型的模块聚合。

下面分别讨论各种类型的模块聚合的含义和特点。

1. 偶然聚合

如果一个模块所要完成的动作之间没有任何关系也没有什么理由而放到一起，或者即使有某种关系，也是非常松散的，则称为偶然聚合。

偶然聚合的最大缺陷在于不易修改，此外，其可读性也极低。

2. 逻辑聚合

如果一个模块内部的各个组成部分在逻辑上具有相似的处理动作，但功能上、用途上却彼此无关，则称为逻辑聚合。

例如，在图 5.25（a）所示的结构中，假设 E、F、G 均为输出报表模块，即从逻辑上讲它们是类似的。如果我们把 E、F、G 全并到一个模块 H 中，则产生了逻辑聚合，如图 5.25（b）所示。模块 H 实际上是逻辑上相似的功能 E、F、G 的简单组合。

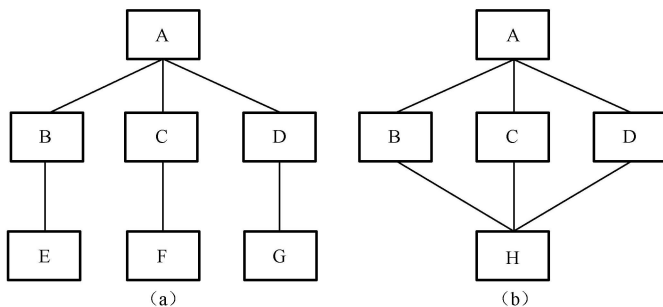


图 5.25 逻辑聚合示例

在调用逻辑聚合的模块时，必须完全知道该模块的内部属性。因此，它与其他模块之间具备相当复杂的耦合，其可修改性差，维护困难。

### 3. 时间聚合

如果一个模块内部的各个组成部分所包含的处理动作必须在同一时间内执行，则称为时间聚合。

时间聚合是将需要同时执行的部分放在同一模块内，如初始化模块需要为各种变量置初值，并同时打开若干个文件，而结束模块则要将变量全部清零并同时关闭文件。

时间聚合模块的紧凑性也较低，其缺陷是修改性较差，维护较困难。但它在一定程度上反映了系统的某些实质，因此时间聚合比逻辑聚合要强一些。

总的说来，以上三种聚合都是很弱的。

### 4. 过程聚合

如果一个模块内部各个组成部分所要完成的动作彼此间没什么关系，但必须以特定的次序（控制流）执行，则称为过程聚合。

过程聚合模块常常是由程序流程图直接演变过来的，其处理动作彼此间并没有什么关系，但在同一控制流的支配下汇集在一个模块中。在过程聚合中，次序是唯一重要的，这种次序可能是顺序、判断或循环。

过程聚合较之时间聚合等要强，但可修改性也不高。

### 5. 通信聚合

如果一个模块内部的各个组成部分所完成的动作都使用了同一个输入数据或产生同一个输出数据，则称为通信聚合。

在图 5.26 中, 模块 A、B 调用 C 时, 使用文件编号作为参数调用, 而 C 则将该文件保存, 并予以打印, 其中保存、打印均是针对同一文件。这里 C 就称为通信聚合模块。

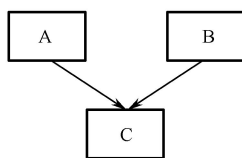


图 5.26 通信聚合示例

通信聚合的模块与其他模块间的联系较简单, 其内部紧凑性比过程聚合强, 但它的各部分的执行次序可以是任意的, 所以容易产生重复的动作。

## 6. 顺序聚合

对一个模块内部的各组成部分, 如果前一部分处理动作的输出是后一部分处理动作的输入, 则称为顺序聚合。

如图 5.27 所示, 模块 A 由两部分组成, 一部分为读入, 另一部分为编辑。显然, 读入部分将其输出数据作为编辑部分的输入, 因此, A 是顺序聚合模块。

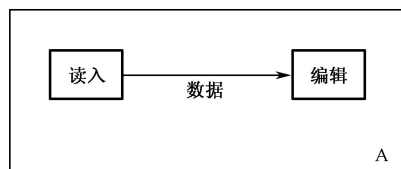


图 5.27 顺序聚合示例

顺序聚合模块的紧凑性比较高, 与其他模块的联系也较低。通常, 根据数据流图划分模块时, 即可得到顺序聚合模块。与通信聚合模块相比, 顺序聚合模块要好一些, 因为无论从数据的角度或执行的先后次序来看, 顺序聚合模块中某一部分的执行完全依赖于另一部分, 但由于顺序聚合模块中可能包含了几个功能, 也可能仅为某个功能的一部分, 因此它比下面将要说明的功能聚合模块要差一些。

## 7. 功能聚合

如果一个模块内部的各个组成部分全部属于一个整体、执行同一功能, 则称为功能聚合模块。

功能聚合的模块都具有一个目的，有单一的功能，因而其界面非常清楚，与其他模块的联系低，可读性、可修改性、维护性、可测试性均很好。另外，许多功能聚合模块可集中为一个模块库，选择其中的一部分便可进行合成，产生新的系统（或子系统）。

功能聚合是最高程度的聚合，在进行模块设计时，应尽可能地追求功能聚合。

下面将从耦合、可读性、可修改性、共用性等方面来对上述七种聚合形式进行比较，见表 5.8。

表 5.8 七种聚合的比较

聚合形式	耦合	可读性	可修改性	共用性
功能聚合	低	好	好	好
顺序聚合	低	好	好	较好
通信聚合	较低	较好	较好	不好
过程聚合	一般	较好	较好	不好
时间聚合	较高	一般	不好	坏
逻辑聚合	高	不好	坏	坏
偶然聚合	高	坏	坏	坏

在设计中，应当尽可能做到模块的高聚合，通过修改设计，使每一个模块执行单一的功能，提高模块的聚合程度，降低模块间的耦合程度，争取获得较高的模块独立性。

#### 5.8.4 若干其他设计原则及有益的建议

模块耦合与模块聚合是模块设计中最为重要的概念，尽可能地降低模块之间的耦合程度和提高模块内部的聚合程度是设计中的两项重要原则。但是机械地照搬这些是远远不够的，甚至会使我们在进行系统设计中走上歧途。除了上述两项原则外，还存在若干辅助性的原则，同时在地不断地反复实践与思考的基础上，人们在信息系统的设计中积累了大量的经验，这些经验对于我们都是极为有益的。下面所介绍的一些原则和建议并不像前面所讲的那样普遍，但也从其他侧面给出启示，并帮助改进系统设计，从而寻找到最佳的设计方案。

##### 1. 改进系统结构

对于初步设计出的系统结构，应该仔细地分析与审查，发现高耦合、低聚合的模块，并通过模块的分解与合并，改进系统结构，从而降低耦合、提高聚合。一般的方法有下述几种。

(1) 对于若干模块共有的一个子功能，应当将其独立抽出，作为一个新的模块

（可被前述的模块所调用）。设计中应提供更多的共用性强的模块，从而简化程序设计工作。

（2）尽可能地采用数据信息作为模块之间联系的媒介。可以通过分解等手段，将一些传递控制信息或其他非数据信息的模块转化为数据耦合的模块。

（3）强调以功能划分模块。每一模块尽量做到只有单一的功能，对于复杂的模块应从功能的角度出发予以分解。

（4）强调系统的整体性大于局部的最优性，不片面追求系统中每一模块的最优设计，局部的优化应服从整体的安排。

## 2. 系统的深度、宽度

系统的深度（Depth）表示系统结构中的控制层数；系统的宽度（Width）则表示控制的总分布，即同一层次的模块总数的最大值。

一般情况下，深度和宽度标志着一个系统的大小和复杂程度，它们之间应有一定的比例关系，即深度和宽度均要适当。深度过大可能说明系统分割得过分细化；宽度过大，则有可能带来系统管理上的困难。

## 3. 模块的扇出与扇入

一个模块控制的直属下级模块的个数称为该模块的扇出（Fan-Out）；反之，一个模块的直接上级模块的个数称模块的扇入（Fan-In）。

在图 5.28 中，模块 A 的扇出系数为 3，模块 F 的扇入系数为 2。

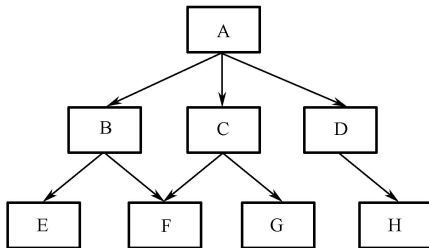


图 5.28 扇出与扇入示例

模块的扇出直接影响系统的宽度，扇出过大意味着该模块的直接下属模块多，控制与协调较困难，也意味着模块的聚合可能较低。这时一般需要增加中间层次的控制模块。扇出过小，则说明上、下级模块或其本身可能过大，应考虑采用分解的方法，使结构变得合理。



模块的扇出系数必须适当。经验表明，一个设计得好的系统的平均扇出通常是 3 或 4，一般不应超过 7，否则出错概率会增大。

模块的扇入通常说明系统的通用性情况，扇入系数越大，表明共享该模块的上级模块数目越多，因而通用性强，维护也较方便，但是片面地追求高扇入可能使得模块的独立性降低。

通常，一个较好的系统结构，高层扇出较高，中间扇出较少，底层模块有很高的扇入。

#### 4. 模块的规模

系统分解为模块时，究竟每一个模块的规模应多大呢？大量的实践表明，一个模块的规模不可过大，也不可过小。过大的模块常常是系统分解不充分，其内部可能包含了若干部分的功能，使模块聚合降低，因此有必要作进一步的分解，即把原有的模块变成若干功能尽可能单一的较小的模块。但分解也必须适度，因为过小的模块有可能降低模块的独立性，造成系统接口的复杂。

#### 5. 模块的作用范围与控制范围

模块的作用范围是指受该模块内部的一个判定影响的所有模块的集合，只要某一模块中含有一些依赖于这个判定的操作，那么该模块就在这个判定的作用范围之内。如果整个模块的全部操作都受该判定的影响，则这个模块连同其上级模块都在这个判定的作用范围内。

模块的控制范围包括该模块本身及其所有的下属模块的集合。模块的控制范围完全取决于系统的结构，它与模块本身的功能并无多大关系。

以图 5.29 为例，模块 A 的控制范围是集合 {A, B, C, D, E, F, G}。

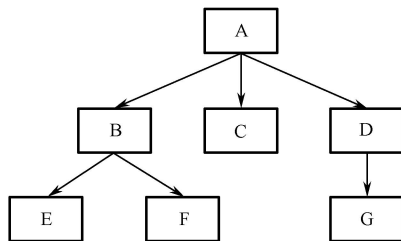


图 5.29 控制范围示例

一个“好”的系统设计，对于模块的控制范围和作用范围，存在着下面两条规则。

(1) 所有受模块 M 的一个判定影响的模块应从属于模块 M，即对任何一个内部存在判定调用逻辑的模块，其作用范围应是其控制范围的子集。

(2) 受模块 M 判定影响的模块，最好局限在模块 M 本身或其直属下级模块，即作出判定的模块与属于该判定作用范围的模块在系统的层次上不能相隔过远，否则会增大模块间的耦合。

现分析图 5.29，假设模块 B 作出的判定影响到模块 E 和模块 C，而模块 C 又不在于 B 的控制范围内，显然这样的结构令人难以理解。为了使模块 B 的判定能影响模块 C，需要设置一个控制标记传递给 B、C 的上级模块 A，再由 A 传递给模块 C，这样就使得模块间的联系增大，同时出现了控制耦合。

设计中，如果发现模块的作用范围不在其控制范围内，可进行下述的改进。

(1) 将作判定的模块合并到其调用模块中或把判定上移到足够高的位置。

(2) 将受判定影响的模块下移到作判定模块的控制范围内。

例如，图 5.29 中的模块 C 可移到模块 B 下，如图 5.30 所示。

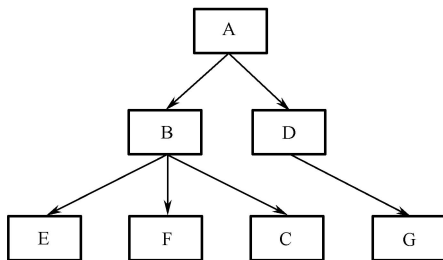


图 5.30 改进的模块结构

## 6. 其他有益的建议

除了上述的一些规则和建议外，在具体设计时，还有一些其他方面的考虑。例如，应该设计单入口、单出口的模块，从而不使模块间出现内容耦合；一个模块，应设计成“暗盒”的形式，只完成一个单独的子功能，这就使得模块内部有很强的聚合；模块的接口设计应简单，若接口复杂，则有可能存在高耦合、低聚合。

以上所列举的一些带有启发性的规则或建议多是从大量的统计中得出的经验，并不是具有普遍性的规律。因此，在设计一个信息系统时，不应生搬硬套，而应本着“具体问题具体分析”的原则，针对问题进行设计，这才是唯一正确的途径。此外，还应善于发现、总结那些对改进系统设计、提高系统质量方面有益的经验，使工作越做越好。

## 5.9 面向数据流的设计

通常所说的结构化设计实际上是一种面向数据流的设计 (Data Flow-Oriented Design, DFOD)，它是与数据流分析相对应的系统设计技术。

面向数据流的设计方法是由 L. Constantine 和 E. Yourdon 等人首先提出的，强调用一组标准的准则和工具来构造、描述系统。它根据数据流图的特性定义两种“映射”，这两种映射能机械地将数据流图转换为模块结构图。

### 5.9.1 结构图

结构图是一种强有力的图形表达工具，可用于表达系统内部各部分的结构和相互关系，是进行系统结构设计的最常用的方法。在结构图中，常用的几种符号如图 5.31 所示。

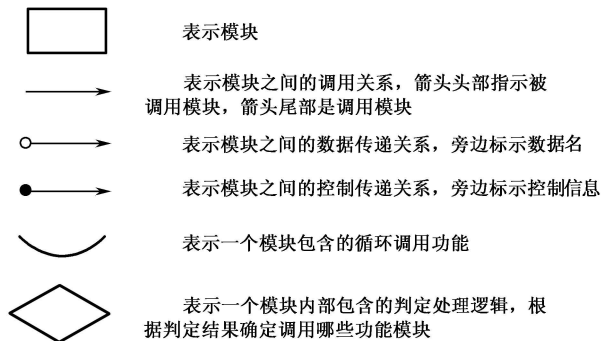


图 5.31 结构图的基本符号

下面通过一个实例来描述结构图的基本画法。

图 5.32 所示是某财务系统中审核凭证处理过程的数据流图。用户向系统输入需要审核的凭证编号，系统从财务数据库中自动读取该凭证编号下的记录，并予以审核，然后显示审核结果。图 5.33 所示就是凭证审核处理过程模块结构图。

需要说明的是，结构图与数据流图有着本质的区别。后者反映的是系统的逻辑模型，是从数据在系统中的流动情况来考虑系统的；前者则是描述系统的物理模型及系统的功能是怎样逐步完成的，它从系统的功能层次上来考虑系统。另外，结构图并没有严格地表示模块的调用次序，而只表明模块的调用关系。虽然许多人习惯于按调用次序从左至右画模块，但结构图并没有这种要求，有时出于减少交叉线或其他方面的考虑，完全可以按别的次序画图。此外，结构图也不指明上层模块在什么时候调用下层模块。

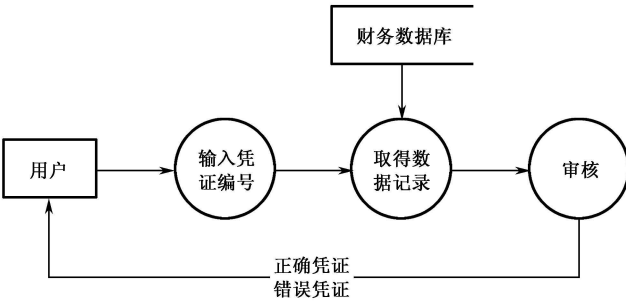


图 5.32 审核凭证处理过程的数据流图

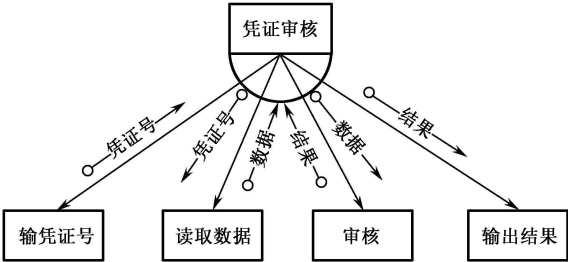


图 5.33 凭证审核处理模块结构图

### 5.9.2 设计过程

下面讨论面向数据流设计的设计过程。首先介绍变换流和事务流两个概念。

#### 1. 变换流与事务流

数据流一般可归纳为两种典型的结构：变换流，事务流。

##### (1) 变换流

在基本系统模型（即顶层数据流图）中，数据通常以“外部世界”所具有的形式进入系统，经过处理后又以这种形式离开系统，如图 5.34 所示。输入数据流沿流入路径进入系统，同时由数据的外部形式变换为内部形式，经系统变换中心加工、处理，作为输出数据流又沿传出路径离开系统，其数据形式又还原为外部形式。

变换流是一种近似的线性结构，可明显地划分为输入、处理和输出三个部分。如图 5.34 所示，数据流分为三段，由数据的外部表示变成内部表示的一段数据流称为系统的输入流；处于系统各种内部变换时间的一段数据流称为系统的变换流；由数据的内部表示又变成外部表示的一段数据流称为系统的输出流。

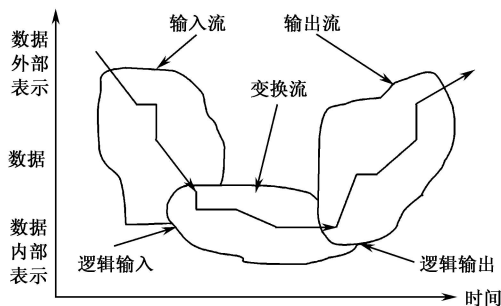


图 5.34 变换流

外部表示的输入（或输出）称为系统的物理输入（或输出），而内部表示的输入（或输出）称为系统的逻辑输入（或输出）。

因此，系统的逻辑输入点与逻辑输出点之间的一段数据流就是系统的变换流；物理输入与逻辑输入之间是输入流；物理输出与逻辑输出之间是输出流。

## （2）事务流

由于基本系统模型呈变换流的形式，因此通常系统中的数据均采用变换流分析。但若数据流具有如图 5.35 所示的形状，则称为事务流。此时，事务（单个数据项）沿传入路径进入系统，由外部形式变换为内部形式后到达事务中心，事务中心根据数据项计值结果从若干动作路径中选择一条继续执行。

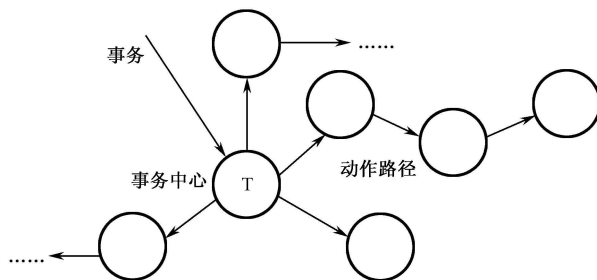


图 5.35 事务流

事务流实际上是以某项业务为基本数据单元来触发事务中心的，并由该事务中心辐射出许多条动作路径，但每次仅沿一条路径流动的数据流。

## 2. 设计步骤

面向数据流的设计方法能方便地将数据流图转换为模块结构图，其过程分为五步：

（1）确定数据流的类型；

- (2) 划定转换的边界；
- (3) 将数据流图映射成基本的模块结构图；
- (4) 分层求精模块结构图；
- (5) 应用模块设计和优化准则，优化模块结构图。

其中，前三步属于基本的设计过程，其结果是得到初始的模块结构图；后两步则是对初始模块结构图的精化。图 5.36 所示是面向数据流的设计过程。

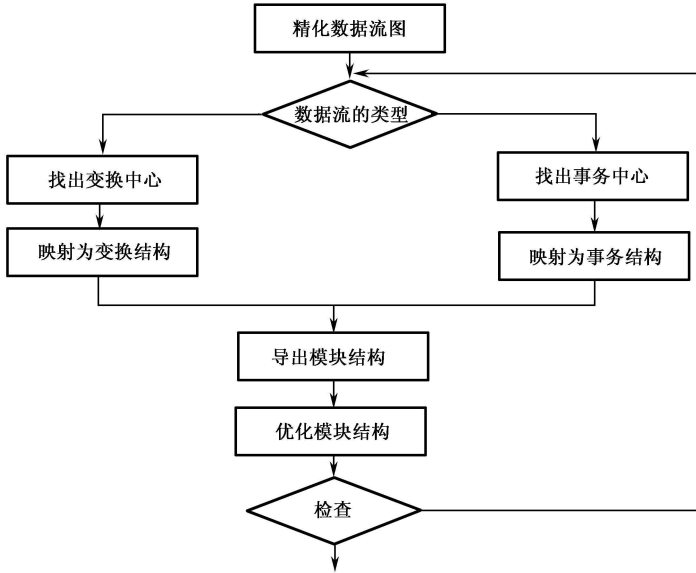


图 5.36 面向数据流的设计过程

通常，一个实际系统的数据流图是变换流和事务流两种类型的混合体。例如，图 5.37 中的数据流从总体上分析属于变换流，但是其主加工部分的数据流图却是事务型的。

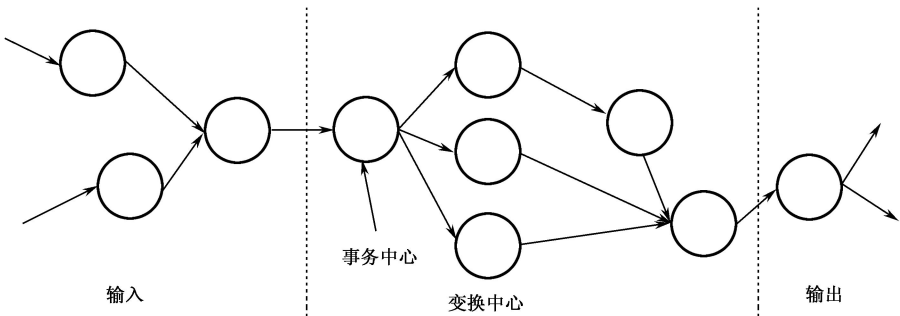


图 5.37 混合型数据流图

### 3. 变换分析

变换分析 (Transform Analysis) 就是从变换型数据流图映射出模块结构图的过程, 也称为以变换为中心的设计。

运用变换分析方法, 首先根据数据流图上的处理框, 找出主要处理功能, 即变换中心, 把数据流图划分为输入、处理和输出三大部分, 从而得到结构图的第一层模块分解图, 如图 5.38 所示。通过对该图做进一步的分解和优化, 便可获得系统的最终模块结构图。

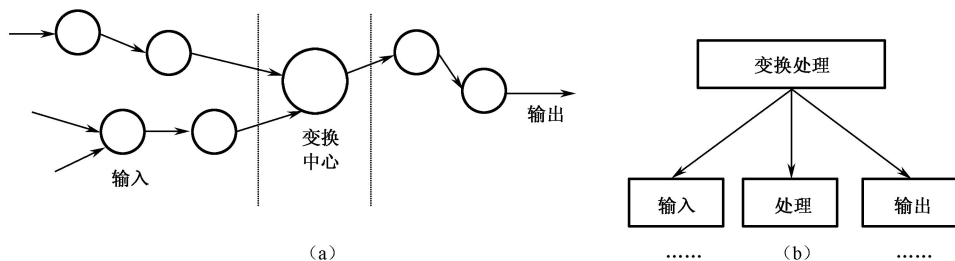


图 5.38 变换分析的一般形式

按照这种策略所产生的模块化系统容易理解、维护, 且具有很高的可修改性。

变换分析方法主要由以下三个步骤组成: 分析数据流图, 确定其主要处理 (变换中心)、输入和输出功能; 设计上层模块; 设计输入、输出和变换中心的下层模块。

#### (1) 确定变换中心、输入和输出

在这一步骤中, 暂不考虑数据流图中的一些支流, 如出错处理等, 而把精力放在主要处理上。

对于一个经验丰富, 且对该系统的系统说明书很熟悉的设计人员来说, 决定系统的变换中心是较容易的。但是, 在设计人员对数据流图不十分了解、变换中心难以确定的情况下, 可以运用下面的方法来予以确定。

第一步, 从物理输入端开始, 沿着每一个由数据源传入的数据流的移动方向进行跟踪, 或者逐步向中心移动, 直到数据流不再被看作系统的输入为止, 这时, 它的前一个数据流就称为逻辑输入。换句话说, 离物理输入端最远的, 但仍可看作是系统输入的那个数据流就是逻辑输入。

第二步, 与第一步中跟踪数据流的方向相反, 从物理输出端开始, 逐步向系统的中心移动, 直到找出离物理输出端最远的, 但仍可看作是系统的输出的那个数据流, 此数据流即为逻辑输出。

第三步，找出系统的变换中心，即位于逻辑输入和逻辑输出之间的处理功能。

图 5.39 所示是根据用户输入的编码修改账目的数据流图的一部分，按上述三步来确定它的变换中心部分。

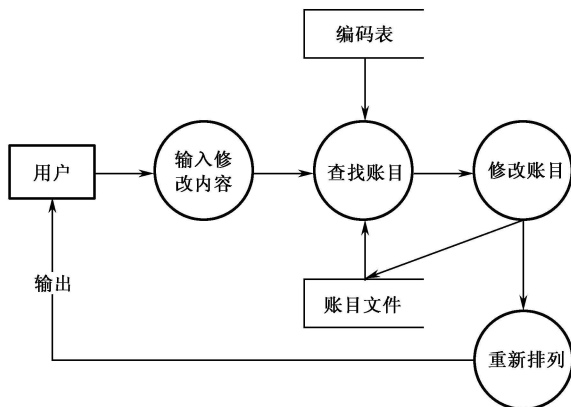


图 5.39 修改账目的数据流图

显然，在图 5.39 中，所有的数据流沿着同一条路径流动。其数据来源有两个，即用户输入内容和原账目数据库记录。在“修改账目”处理过程前的数据流显然均属于输入数据流。“修改账目”处理过程流出两条数据流，一条重新写账目数据库，另一条则将结果反馈给用户。因此，可以确定出变换中心“修改账目”处理过程。图中不必考虑“没检索到”这样的非主要数据流。

## （2）设计上层模块

自顶向下设计的关键是找出“顶”的位置，确定了变换中心实际上就决定了系统结构的“顶”。系统的上层模块分两层，顶层是一个主模块，主模块的下层（第一层）由输入部分、变换中心部分和输出部分组成。可以这样设计结构的第一层：为每一个逻辑输入设计一个输入模块，其功能是向主模块输入数据；为每一个逻辑输出设计一个输出模块，其功能是将主模块提供的数据输出；对变换中心部分中的每个变换设计一个变换模块，其功能是接收输入数据，进行变换，然后输出。图 5.40 所示是在对图 5.39 进行分析的基础上，画出的第一张结构图。

这样就得到了上层结构，最高层模块“修改账目”起控制和调度作用，一般说来，它根据一些逻辑判定来控制对下层模块的调用。

## （3）设计输入、输出和变换中心的下层模块

设计下层模块的工作实际上是自顶向下、逐步细化上层模块的过程。这里，要充



分运用前面已讲述过的模块设计原则和经验,如模块间的耦合度、模块内部的聚合度、分解、扇出和扇入、控制范围和判定作用范围等,合理分解和组织系统结构,必要时采取适当的方法改进系统的结构。

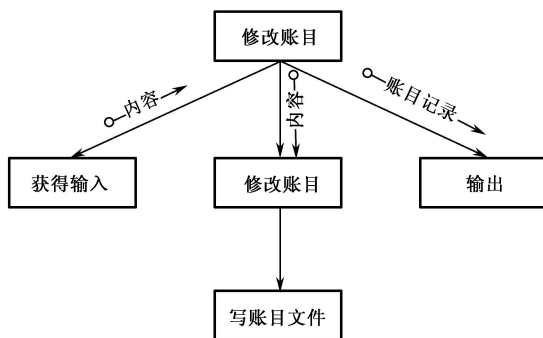


图 5.40 修改账目的初始结构图

设计输入、输出部分的下层模块,一般可按下述方法进行。

由于输入模块是向其调用模块提供数据的,因而它本身必有一个数据来源。另外,输入模块必须将这些数据按调用模块所需的形式进行变换后,才能提供给调用模块。因此,输入模块可由两部分组成,一是接收输入数据;二是将这些数据进行变换。这样,可为每一输入模块设计两个下层模块,即数据接收模块和变换模块,如图 5.41 (a) 所示。

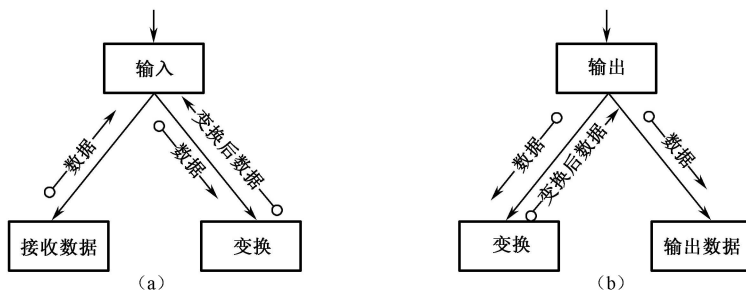


图 5.41 输入、输出部分的下层模块

同样,输出模块也由两部分组成,一部分是将调用模块提供的数据变换成输出的形式,另一部分是输出数据,如图 5.41 (b) 所示。

上述过程可以自顶向下、递归地进行,直到达到系统的物理输入端或输出端为止。设计变换模块的下层没有一定的规则可遵循,一般应仔细研究数据流图中相应的

主要变换部分，应用模块设计原则来考虑每一变换的分解。

图 5.42 所示是在图 5.40 基础上画出的比较完整的结构图。

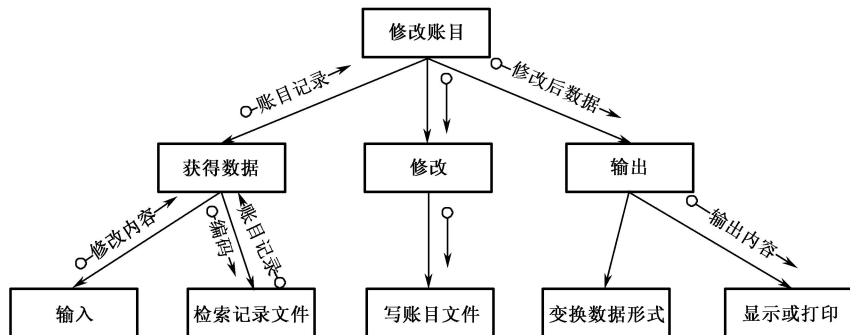


图 5.42 修改账目初始结构图的改进

#### 4. 事务分析

根据前面的分析知道，如果一个系统从同一个数据源进入数据，而且数据流流动的逻辑路径是相同的，则采取变换分析的方法进行设计是合适的。但是，如果进入系统的数据或事务有若干种，而且需要采取不同的处理方法，那么变换分析的设计策略就不适用了，这时就要采取事务分析的方法来设计系统。

一件事务实际上是指一组数据或事件流入系统，并引起一组处理动作。事务分析（Transaction Analysis）就是对事务型结构的数据流图进行变换，从而导出标准的结构图的一种方法，它是结构化系统设计中另一项主要的设计策略。事实上，一个大的系统一般是变换型结构和事务型结构的混合体，所以往往需要同时采用变换分析和事务分析。通常是以变换分析为主、事务分析为辅进行设计的，即：

- （1）找出输入、输出及变换中心，设计系统结构图的上层；
- （2）根据数据流图各部分的结构特点适当地运用变换分析或事务分析方法，得出初始模块结构图；
- （3）优化处理，导出系统最终的模块结构图。

事务分析像变换分析一样，采用自顶向下、逐步分解的方法，即先设计主模块，再为每一个事务设计一个事务处理模块，然后为各个事务处理模块设计下层的操作模块，最后为操作模块设计下层的细节模块。假定事务型数据流图如图 5.35 所示，则事务分析的一般变换过程如图 5.43 所示。

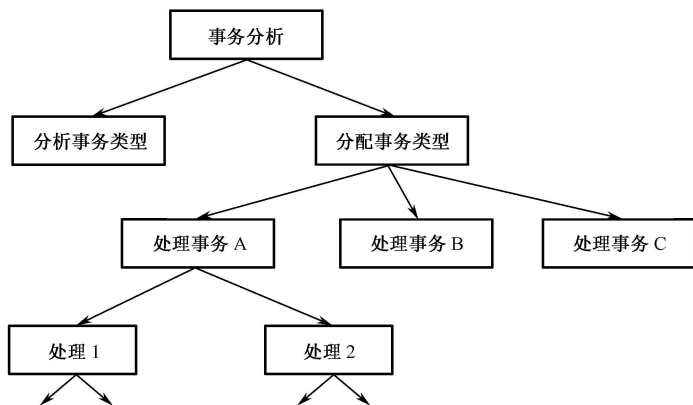


图 5.43 事务分析的一般变换过程

一般情况下，事务分析具有如下的步骤。

(1) 识别事务源：通过数据流图和数据字典，正确识别各种事务及其类型。大部分的事务在数据字典中已有定义，但有时某个事务可能是系统的输入、变换中心和输出中产生的，此时只有在对变换型结构的系统进行上层设计之后，才能得到正确的识别。

(2) 确定适当的事务型结构：根据模块的设计规则，建立适当的事务型结构。

(3) 确定各种事务及相应的处理：通过数据流图和数据字典，一般可找到这方面的信息，但对系统内部产生的事务要仔细地找出每一处事务及其处理动作。

(4) 合理建立公用模块：在事务分析过程中，有时若干个模块存在相同的处理动作，把它们独立出来成为一个共同的低层模块，即建立公用模块。

(5) 对每一类型的事务，建立一个事务处理模块：在强调高聚合的前提下，把系统中十分相似的某些事务组合起来成为一个模块，但决不允许出现逻辑聚合的模块。

(6) 对事务处理模块，建立直属于该事务处理模块的下级模块（操作模块），分解中要符合模块的设计原则，同时注意公用模块的设计。

(7) 为操作模块设计其全部的细节模块，必要时可进行这一步的设计，同时要尽可能地使用公用细节模块。

下面通过一个示例来介绍事务分析的过程。

在企业销售子系统中，常常需将信息分门别类地加以整理，如按日期、按地区、按顾客、按产品进行销售统计，以便进行销售分析等。图 5.44 所示就是销售分析的数据流图。

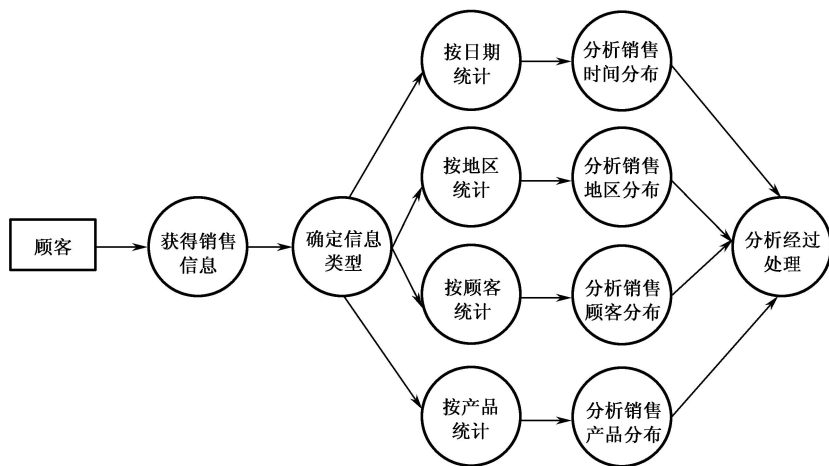


图 5.44 销售分析的数据流程图

经分析可知，这一流程图很明显是一事务型结构，同时不难确定事务源为“销售信息”，且共有“按日期统计”、“按地区统计”、“按顾客统计”和“按产品统计”四个事务，其操作分别为相应的销售分析。于是，可得出如图 5.45 所示的事务型结构图。

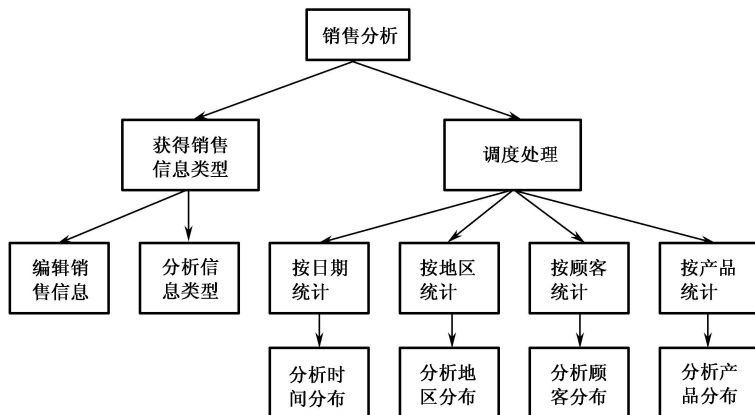


图 5.45 销售统计分析结构图

上述转换到的结构图并不是唯一的，完全可以根据图 5.44 所示的数据流程图得出其他形式的系统结构图。

### 5.9.3 设计优化

初始结构图的导出,并不意味着系统模块设计的完结,还存在着大量需改进的优化工作,这些工作的出发点就是已讨论过的设计原则和有益的经验。值得注意的是,过分强调系统的最佳设计而不考虑系统的可运行性并不是可取的。因此,设计人员应致力于研究能够满足所有功能和性能要求,而且符合系统设计原则和经验的设计方案,并从系统的角度出发,尽可能地使全局达到优化,即首先应当使系统能进行工作,然后再想方设法地使系统的运行更快、更省。

设计人员应该力求系统结构的简单,因为结构简单通常意味着设计合理、系统效率高,因此要做到在有效的模块化的前提下使用最少量的模块,以及在满足信息需求的前提下使用最简单的数据结构。

改进初始设计的基本原则仍然是降低模块之间的耦合及提高模块内部的聚合,一般有两个步骤,如下所述。

#### 1. 检查初始设计方案

在仔细研究初始设计方案的前提下,着重检查以下几个方面。

(1) 系统的结构:系统的深度与宽度,模块的扇出、扇入,以及模块的作用范围与控制范围等是否存在不合理的现象。

(2) 模块之间的耦合度:模块之间联系的方式,应满足低耦合的要求;模块的接口是否清晰、简单。

(3) 模块内部的聚合度:每一个模块的功能应清楚,其内部紧凑性应高于通信聚合,输入、输出表达明确。

(4) 系统的性能:系统是否具有较强的可读性、可维护性、可修改性及可靠性;系统与用户之间的接口是否简单、明确、易于理解;系统能否实现,能否正确地工作。

针对上述四个方面的检查,基本上可查出系统初始设计方案中的错误,从而改进设计,提高系统质量。

#### 2. 解决的办法

系统改进的过程具有很强的试探性,如何对各种可能的方案作比较和权衡,除了基本的设计原则外,设计人员的经验是非常重要的。

改进系统设计仍然是从两个方面着手,即如何降低模块之间的耦合和提高模块内部的聚合。这两个问题在前面已作过详细的介绍。设计人员应从实践中不断总结、认

真体会模块设计的基本原则和一些有益的经验,并合理地加以运用,只有这样才能使设计出来的方案行之有效。

通常,应注意下列四个方面:

- (1) 具有最佳设计的,但不能工作的系统是糟糕的;
- (2) 系统全局的优化远远超过局部的优化;
- (3) 简单而又能满足所有功能、性能要求的设计方案,意味着系统实现方案运行高效、可靠性高;
- (4) 可读性、可维护性、可修改性好的系统具有更强的生命力。

## 习 题

1. 系统设计的原则有哪些?
2. 试述系统设计的步骤。
3. 简述结构化系统设计的思想和特点。
4. 数据流图的作用是什么?它有哪些基本成分?
5. 数据字典的作用是什么?它有哪些基本词条?
6. 分别用决策树和决策表表达下面的航空行李托运费的算法。

按规定:重量不超过 30 千克的行李可免费托运;重量超过 30 千克时,对超运部分,头等舱国内乘客收 4 元/千克;其他舱位国内乘客收 6 元/千克;国外乘客收费为国内乘客的 2 倍;残疾乘客的收费为正常乘客的 1/2。

7. 某工厂人事部门对一部分职工重新分配工作,其分配原则如下:

如果年龄不满 18 岁,文化程度是小学,则脱产学习;文化程度是中学,则当电工。如果年龄满 18 岁但不满 40 岁,如果文化程度是小学或中学,若是男性,则当钳工;若是女性,则当车工;若文化程度是大学,则当技术员。如果年满 40 岁及以上者,文化程度是小学或中学,则当材料员;若文化程度是大学,则当技术员。

用决策表表达该分配问题。

8. 假设某考务系统要求有如下功能:

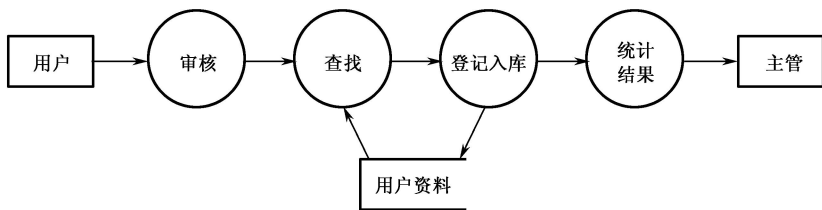
- (1) 对考生送来的报名单进行检查,不合格的退回;
- (2) 对合格的报名单编好准考证号后将准考证送给考生,并将汇总后的考生名单送给阅卷站;
- (3) 对阅卷站送来的成绩单进行检查,并根据考试中心制定的合格标准审定合格者;
- (4) 制作考生通知单(含成绩及合格/不合格标志),送给考生;
- (5) 按地区进行成绩分类统计和试题难度分析,产生统计分析表。

请绘出该系统的两层 DFD。

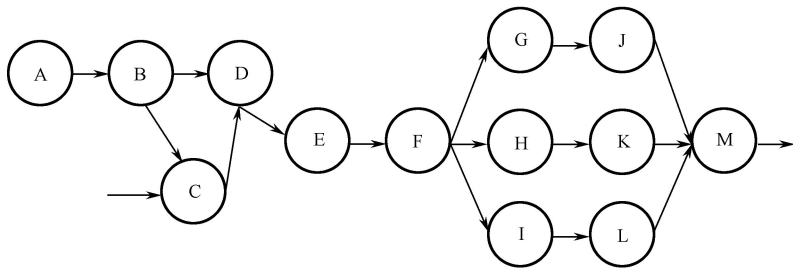
9. 以下是某学校教材购销系统的要求,请用 DFD 表达该需求。

- (1) 根据学校的教学计划,向选课的学生及时供应所需的教材。要求审查学生购书单的有效

- 性，对有效书单并且有库存的发售教材，对暂时缺货的教材进行缺货登记；
- (2) 根据缺货登记补充采购所缺的教材，通知学生补购。要求将缺货登记表汇总为缺货单，由书库管理员采购，待购教材到货后，及时通知学生补购。
10. 设某图书馆管理系统有如下功能要求，请画出该系统的两层 DFD。
- (1) 购入新书：对新书，编目录索引卡片和唯一流水号，并写入图书目录文件。
- (2) 读者借书：读者填借书单，管理员检查借书单的有效性，无效拒借；借书单有效，检查读者借书总数是否超数？超数也拒借；未超数者可借，并将借书情况写入借书文件中。
- (3) 读者还书：根据图书目录和流水号，从借书文件中读出与该书相关的借阅记录，写入还书日期，同时检查是否超期，超过期限罚款，给读者罚款单，并填写罚款记录。
11. 在结构化分析和设计中，主要采用了哪些分析、设计工具，这些工具之间有什么联系？
12. 系统设计阶段主要包括哪些方面的设计？
13. 什么是变换分析？说明变换分析的特点和步骤。
14. 试从下面的数据流图 (DFD) 中得出相应的模块结构图。



15. 将以下 DFD 图转换为模块结构图。



## 第 6 章 面向对象系统分析与设计

随着科学技术的发展，信息系统的分析和设计方法也在进步。继 20 世纪 70 年代结构化分析和设计方法发展成熟之后，80 年代出现了面向对象的技术和方法。到目前为止，面向对象的技术和方法在信息系统开发的分析、设计、实现、测试等各个阶段得到了全面应用。事实表明，面向对象技术和方法的应用大大提高了信息系统的开发速度和质量，使信息系统的开发达到了一个崭新的阶段，同时也表明信息系统的开发将从结构化方法向面向对象的方法转变。

### 6.1 面向对象的基本概念

要了解面向对象的分析和设计方法，首先需要了解面向对象的基本概念。

#### 6.1.1 对象

客观世界是由客观事物和事物之间的相互关系构成的，在不同的方法中，有时称这些事物为实体，有时称为客体等。而在面向对象中，把客观世界中存在的事物看作对象，对象是构成客观世界的基本单位。

对象是用来描述客观存在的事物的，它是构成系统的基本单位，是对客观世界中事物的抽象描述。它可以是有形的（如一栋房子），也可以是无形的（如一项实施计划）。对象的特征通常包括动态特征和静态特征。静态特征是可以由某种数据描述的特征，动态特征是对象所表现的行为和具有的功能。一个对象由该对象的一组属性（也可称为数据）和对这组属性进行操作的一组行为（也可称为功能、方法、服务）组成。面向对象强调对象是由属性和行为构成的封闭整体，它向外界提供了一组接口界面，外界通过这些接口与对象进行交互，而对象的具体实现方法对外界而言是封闭的。这一点很好地体现了信息隐蔽的原则。

按照面向对象的观点，既然客观世界中存在众多对象，那么如何来区分这些对象是首先必须解决的问题。在现实中，人们一般通过命名的方法来识别不同的事物，同样，区分不同的对象也可以采用命名的方式来解决。我们把对象的名字称为对象标识。任何一个对象都有一个唯一的对象标识存在，相同的对象标识是同一对象，不存在名字相同的两个不同的对象。



对象与其他事物相似，都有一个产生、存在和消亡的过程，我们称之为对象的生命周期。在现实中，对象的生命周期往往是一个自然消亡的过程，而在以计算机为基础的信息系统中，由于受到物理设备容量、速度及软件等方面的限制，因此对象的生命周期应尽可能限制在其需要的范围以内，以提高资源的利用率。

需要说明一点，“对象”一词在使用上有时是指某个具体对象，有时是对象概念的泛指。例如，面向对象就是泛指对象，是对象含义的泛化，不具体说明某一个对象，而是所有对象概念的统称；如果说系统中有某某对象，则指某个具体对象，即特指某个对象本身。这种不同的用法通过上下文可以容易地辨别。

### 6.1.2 消息

信息系统的功能需要其中的许多对象的行为有机组合才能完成。如何要求对象完成一定的处理工作？对象之间如何进行联系？所有这些是通过对象之间的消息通信来完成的。当系统中的其他对象请求这个对象执行某个功能时，它就响应这个请求，完成相应的功能。在面向对象当中，把对象发出的服务请求称为消息。消息是对象之间进行通信的数据结构，对象之间通过传递消息进行联系，消息统一了控制流和数据流。例如，在售票窗口告诉售票员买一张火车票，这就是一条消息，它向售票员发出服务请求，售票员接收到消息以后完成相应的售票服务。这条消息包括了：消息接收者（售票员），要求的服务（买车票），输入信息（车次、时间、票价），回答信息（车票、找回的零钱）。下面是消息的定义。

消息就是向对象发出的服务请求，它应含有提供服务的对象标识、请求的服务、输入信息和回答信息。

对象对外提供的每一项服务需要规定接收消息的格式，这种规定称为消息协议。

### 6.1.3 类

将众多的事物进行归纳而划分成不同的一些类是人们认识客观世界的基本思维方法。这种方法依据的原则是抽象，它忽略了客观事物的非本质特性，将客观事物所具有的共同点，即本质特性划分为一类，形成对事物的本质认识，得出事物的抽象结果。例如，房子、树木等概念，它们并不具体指某一个事物，而是一类客观事物的抽象概念。在面向对象中，类的概念就是上述抽象思维的反映。所谓类，就是具有相同属性和行为的一组对象的集合，它为该类的全部对象提供了统一的抽象描述，其内部包括类的属性和类的行为两部分。

类与对象之间的关系好比模具与用这个模具铸造出来的铸件之间的关系。类给出

了属于该类的所有属性和行为的抽象定义，而每个对象是符合这种定义的一个具体实体。对象是实实在在存在的，而类仅仅是事物的抽象概括，它本身并不实际存在。因此，也将对象称为类的一个实例，这个过程称为实例化。

在面向对象的程序设计中，允许定义类（Class）。如同其他数据类型（如整型、实型）一样，类可以定义相应的变量，这个变量就是对象（类的实例）。从数据结构的角度来看，类是抽象数据类型（Abstract Data Type）的实现，它能充分体现信息隐蔽的原则。

#### 6.1.4 继承

前面讨论了类的概念，在类与类之间的关系中，继承关系是最重要的关系之一。没有继承关系的类是一种平坦结构，它不允许类之间实现信息的共享，从而不能简化复杂问题的处理。有了继承关系、子类继承父类的属性和方法，就可以有效地控制问题的复杂程度，可以在不同的继承级别处理相应的问题，从而使得一个复杂的问题可以在逐级的抽象当中清晰地描述出来。

所谓继承（Inheritance），是自动共享类、子类和对象中的属性和行为的机制。被继承的类称为父类（基类/超类/一般类），继承的类称为子类（派生类/特殊类）。继承意味着子类“自动拥有”或者“隐含地复制”父类中的属性和行为，在父类中已经定义的行为和属性不必在子类中再定义，子类自动拥有父类的属性和行为。为了能更加清楚地理解什么是继承，下面用一个简单的例子加以解释。

当类 B 继承类 A 时（如图 6.1 所示），表明类 B 是类 A 的子类，而类 A 是类 B 的父类。类 B 由两个部分组成：继承部分和增加部分。继承部分是从类 A 继承来的，增加部分则是类 B 所专有。

在实际中，采用继承方法可以非常清晰地描述出复杂的问题。其方法与人们认识问题的方法也是相似的。例如，在自然科学中，经常采用逐层分类的方法来认识问题。又如，在生物学中，可以将生物描述为以下继承关系，如图 6.2 所示。其中，动物类和植物类都属于生物类，它们继承生物类中的特征，自身也有其自己的特征，而脊椎动物类和软体动物类继承了动物类的特征又有自己的特征，等等。

子类仅从一个父类中继承属性和行为，称为单重继承，否则称为多重继承。多重继承的子类同时继承其所有父类中的属性和方法。例如，在图 6.3 中，类 C 继承类 A 和类 B 的属性和方法。

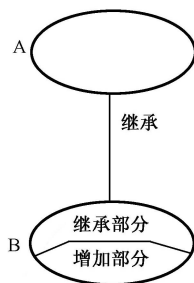


图 6.1 继承

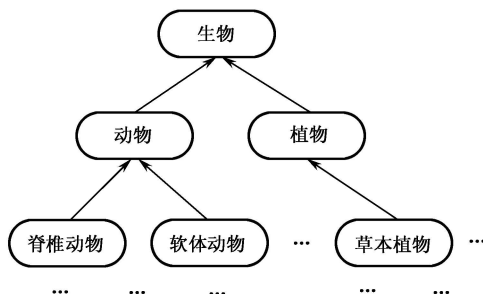


图 6.2 继承实例

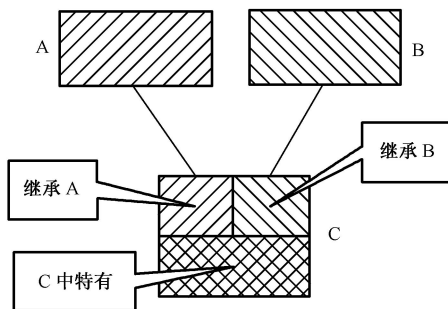


图 6.3 多重继承

在客观世界中，多重继承的关系比比皆是，使得类和类之间的关系变得错综复杂。在面向对象中，一方面，多重继承可以很好地描述客观事物间的继承关系；另一方面，面向对象程序设计语言出于技术和效率的考虑往往对多重继承作一些限制，比如，C++中允许多重继承，但是需要程序员非常清楚不同类之间的继承关系，避免形成继承回路等；在Java等其他一些语言中，仅允许单重继承。

### 6.1.5 封装

封装是面向对象方法的一个重要原则。所谓封装，就是把对象的属性和行为相结合构成一个独立的基本单位，并尽可能地隐蔽对象的内部细节。它有两方面的含义，其一是对象和属性的结合，强调属性和行为的一体化，二者不可分割；其二是信息隐蔽，即尽可能地隐藏对象无须为外界所知道的属性和行为的细节，形成一个对外的屏障，仅仅向外界提供有限的属性和行为（也称为对象接口）。这样，只要对外的接口不

发生变化，其内部的变化对外界产生的影响非常小。

在现实中，许多事物都体现了封装的思想。例如，汽车的方向盘具有控制方向的作用，至于控制方向的具体技术方法却可以完全不同，可以采用机械控制方式，也可以采用液压控制方式，还可以采用电传控制方式等。只要方向盘的功能、作用和使用方法没有发生变化，采用何种控制方式对驾驶者而言都不重要，即产生的影响很小。事实上，在信息系统中，当需求变化时，系统中最容易变化的部分是功能；其次是与外部系统或设备的接口；第三是描述问题域事物的数据；最稳定的是对象。面向对象方法对变化有适应性，得益于封装。它以最稳定的对象作为构成系统的基本单位，把最容易变化的属性和行为封装到对象当中，变化仅仅影响对象内部，只有通过有限的对外接口才对外部产生影响，从而有效地抑制了变化所带来的“波动效应”。

与封装相对应的是可见性，它是指对象属性或方法允许被存取和引用的程度。封装无疑会带来好处，但如果强调严格封装，对象内部的任何属性和行为都不允许进行访问，那么对象需要增加许多仅负责一些简单处理的行为，仅允许通过消息来服务，这样实现起来将复杂得多。因此，对象需要一定的可见性，需要在封装性和可见性之间找到一个平衡点。目前，多数的面向对象的程序设计语言都允许定义属性和行为为外界所见。

### 6.1.6 多态

多态可以简单地理解为一个动作具有多种表现形态。多态的特征在现实世界中同样普遍存在，只不过早期没有为人们所注意而已。例如，绝大多数植物都会开花，对于开花这个普遍行为，对不同的植物其表现的形态是多种多样的，有的白天开，有的晚上开，有的向上，有的向下，可以开出绚丽的花朵，可以开出千姿百态的形状。既然开花这样一个动作在每个不同的植物上的表现方式可以不同，说明开花这个动作具有不同的形态。再如物质的运动，如行星的运动、海洋的运动、植物的运动和动物的运动等，其具体的运动方式、形态可以完全不同，这说明运动也可以存在多种形态。

支持多态是面向对象程序设计语言的基本特征之一，它有助于对多态含义的理解。例如，图 6.4 所示是几何图形的继承关系。在根类“图”中定义了一个服务“绘图”，继承根类的其他子类都拥有“绘图”的功能。对于不同的几何图形，“绘图”产生的结果完全不同，椭圆对象画出椭圆，圆对象画出圆，多边形对象画出多边形，等等。然而，所有的类都是执行一个相同的“绘图”功能来完成的。这时，“绘图”功能具有多种形态，尽管它们动作的名字相同，但是不同的对象表现出来的行为完全不同。

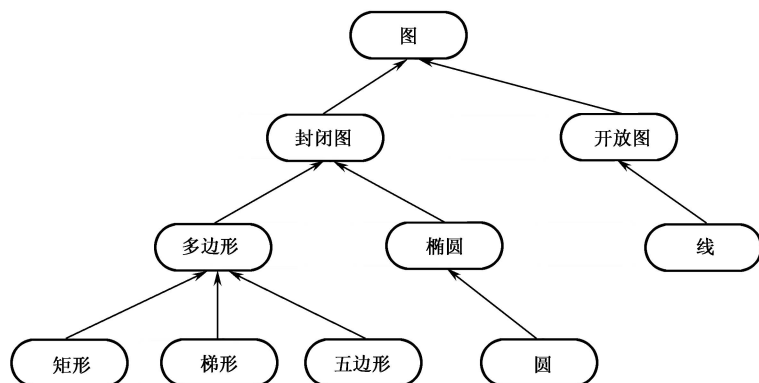


图 6.4 多态

所谓多态 (Polymorphic)，就是一个名字具有多种语义。这种说法是广义上多态的含义，然而，在面向对象中，还需要对多态作进一步的规定，即在具有继承关系的对象或类中，子类继承父类的某个行为，同时，在子类中这个行为的表现又具有其特点，表现形式不完全相同，那么这个行为就具备多态特征。

在面向对象程序语言中，与多态相关的语言功能有重载、动态绑定和类属。

(1) 重载 (Overload)：子类中对父类的服务或功能进行重新定义。

(2) 动态绑定 (Dynamic Binding)：在运行时刻根据对象接收的消息来确定连接运行哪一段代码。

(3) 类属：对象的行为（服务、功能）参量的类型可以是参数化的。

## 6.2 面向对象的方法论

面向对象是一种思想、方法，是人们认识客观世界的一种方法论。“对象” (Object) 一词的出现最早可以追溯到古希腊哲学家的论著中，其含义是表达以客观对象为中心看待哲学问题的观点。在信息技术界，面向对象方法是一种把面向对象的思想应用于信息系统建设过程中、指导开发活动的系统化方法。

### 6.2.1 面向对象的发展过程

在计算机技术的发展过程中，普遍认为面向对象的概念来自挪威 K. Nyguard 等人开发的用于模拟离散事件的程序设计语言 Simula 67。该程序设计语言与以往语言的主要不同在于：从一个新的角度描述与理解客观事实，并首次在程序设计中将数据和与

之对应的操作结合成为一个整体，提出封装（Encapsulation）的概念。尽管 Simula 67 不是真正的面向对象的程序设计语言，但它的思想方法和处理方法已经有了面向对象的痕迹，从此“面向对象”正式登上了历史舞台。

真正的面向对象程序设计（Object Oriented Programming, OOP）语言是由 Alan Keyz 主持设计的 Smalltalk 语言。在 Smalltalk 中，无论整型、实型还是复合型数据结构，它们都是对象。对象具有属性、行为、方法和消息，具有封装、继承和多态等特性。面向对象的概念也是在 Smalltalk 中最先被明确提出的，由此奠定了面向对象的基础。

随着时间的推移，面向对象越来越被人们广泛接受。进入 20 世纪 80 年代后，Xerox 公司推出了 Smalltalk-80，并引起人们的广泛重视。与此同时，Bell 研究所的 B. Stroustrup 着手在 C 语言的基础上加以扩展，使之成为一个面向对象的程序设计语言，并定名为 C++。虽然，C++ 语言不是纯粹的面向对象的语言，但它却继承了 C 语言容易学习、不需要特殊计算平台的特点，这些明显的优点，受到计算机界的普遍欢迎，之后许多 C++ 商业版本和开发工具相继问世。C++ 的出现，促进了面向对象技术的发展，也使它本身有可能成为标准化的面向对象程序设计语言，ISO 也为此成立了一个工作小组（ISO/IEC JEC1/SC22/WG21）。除了 Smalltalk-80 和 C++ 外，其他一些面向对象的语言也在 80 年代相继出现，如 Objective-C、Object Pascal、Common Lisp、Eiffel 等。这时面向对象技术进入了蓬勃发展的时期。

促使面向对象技术兴起的主要原因如下所述。

（1）微电子技术的迅速发展，使计算机性能不断提高，价格不断下降，软件向高质量、图形化、工具化和集成开发环境的方向发展，从而推动了面向对象的发展。

（2）信息系统的规模不断扩大，复杂程度日益提高，原有的方法已很难满足要求，所以需要新的方法在多个层次上进行抽象，以满足应用的需要，面向对象的方法恰恰可以满足这种要求。

（3）新的工程技术的发展，如多媒体技术、CAD、CAM 等，需要描述许多复杂的事物，而面向对象的方法是描述这些复杂事物的最佳选择。

（4）软件的发展远远落后于硬件的发展，难以满足应用的要求，因此迫切需要有新的开发过程模型和新的方法论支持复杂信息系统的建模。面向对象技术在这方面提供了一个全新的方法。

80 年代后期，面向对象程序设计语言已经发展并达到了比较成熟的阶段，同时面向对象的应用领域也在迅速扩大。一方面，将面向对象方法应用于信息系统开发方法中，出现了面向对象的系统分析和设计方法，从方法论上彻底摆脱了传统的结构化系统分析和设计方法的束缚，使信息系统开发进入一个全新的阶段；另一方面，面向对

象技术也在计算机等相关专业领域得到发展，如面向对象数据库、面向对象软件开发环境、面向对象的操作系统等。

### 6.2.2 分析方法的改进

在面向对象方法产生以前，出现了多种分析、设计方法，其中以功能分解方法、数据流方法和信息建模方法最有影响，下面就这三种方法与面向对象的方法进行比较。

#### 1. 功能分解方法

功能分解方法从系统功能的组成和分解的角度来组织系统。它首先确定整个系统的功能，之后再一步步地分解为若干子功能，同时需要定义子功能之间的通信接口，直到这些功能可以给出明确的定义为止。

功能分解方法的出发点是直接从用户提出的功能要求的角度来认识问题域中的事物的。由于这种方法是用户所需要的最直接、最易于表达和理解的方式，同时分析、设计人员也最容易从用户那里得到所需的第一手资料（如统计报表、工作流程等），因此开始进行分析工作是比较容易的。然而，随着分析工作的深入，子功能和接口逐渐增加，复杂性程度大大增加，对系统的把握随分析的深入也愈渐困难。虽然这种方法易于上手，但是它只是从系统表现出来的功能角度出发，而忽略了问题域中的本质事物，因此，功能分解方法构成的系统只是真实系统的间接映射。另一方面，这种方法随需求变化的适应性能力也较差，由于系统随着外界环境的变化而发生相应的变化，也可以由内部的因素而发生变化，这些变化的直接反映就是功能的变化，因此，功能是系统中最不稳定的因素。恰恰由于功能分解方法从功能的角度出发来分析、设计系统，所以使得基于功能的数据结构也会随着系统功能的改变而同时改变，从而很难适应需求的变化。最后，功能分解方法得到的功能之间的接口的范围很宽，缺少必要的控制和明确的限制，因此，不同的分析、设计人员得到的系统的功能接口可能差异很大，这使得局部出现的错误和局部的修改容易产生全局的影响。

功能分解方法较好地运用了过程抽象的原则，即通过系统表现出来的功能来反映问题域中的事物。面向对象的方法也借鉴了它的优点，只不过面向对象方法将过程抽象的原则限制在对象之中，也就是对象自身应具备的功能，具体通过定义对象的服务（功能）方式完成。这样明确了功能仅仅是对象所具备的功能（或提供的服务），由于系统中对象是稳定的，系统功能的变化反映为对象功能的变化，因此，对象功能的变化所产生的影响也仅仅限制在对象内部，局部的修改和变化也只涉及对象本身，其波动影响较功能分解方法要小得多。



## 2. 数据流方法

数据流方法又称为结构化方法，它将系统映射为数据流和处理过程，从系统中数据的流动和数据的处理环节入手进行分析和设计。它由数据流、数据存储、数据处理的字典等组成，其中每一项反映数据的不同方面，从而形成系统的全貌。数据流方法相对功能分解方法严格，数据流从源点到终点的过程中，允许对数据进行变换、存储和传输，不允许凭空产生和消失，同时按照抽象到具体的分解方法，逐层求精。这种严格性可以避免许多错误和疏漏。

与功能分解方法比较，数据流方法强调问题域的研究，是从问题域中的数据的流动、处理和存储的角度描述系统的，与功能分解方法一样是问题域中事物的间接映射。当问题域复杂时也很难检验分析的正确性，当功能发生变化时，与之相关的多个数据流需要修改，进而影响其他处理，因此，数据流方法适应变化的能力较弱。

数据流方法在分析和设计阶段之间存在较大的差异。分析阶段采用数据流，设计阶段采用模块，二者之间在表述上不一致，也没有一种严格的、可转换的规则，因此从分析到设计的转换比较困难。

与数据流方法相比，面向对象的方法具有以下一些特点。

(1) 面向对象方法强调把问题域的事物直接映射为对象，符合人们通常的思维方式，减少了数据流方法从问题域到分析的映射误差。

(2) 面向对象方法从分析到设计再到编码采用一致的模型表示，后一阶段可以直接复用到前一阶段的工作成果中，弥合了数据流方法从数据流图向模块结构图转化的鸿沟。

(3) 面向对象方法把属性和行为封装在“对象”中。当其功能发生变化时，保持了对象结构的相对稳定，使变动局限于一个对象的内部，减少了改动所引起的系统的波动效应。所以，面向对象方法具有易于扩充、修改和维护的特性。

(4) 面向对象方法具有的继承性和封装性支持软件复用，并易于扩充，能较好地适应复杂大系统不断发展和变化的要求。

## 3. 信息建模方法

信息建模方法 (Information Modeling) 是由实体—关系 (Entity-Relationship) 方法发展而来的，1981 年 M. Flavin 进行改进并称为信息建模方法，1988 年由 S. Shlaer 和 S. Mellor 发展成为语义数据建模方法并引入了面向对象的特点。

信息建模方法的发展与数据库设计有很深的渊源，其中实体—关系模型、语义数据模型都与信息建模方法相关，但本质上它是一种分析方法，即从问题域中分析映射



得到相应的模型。信息建模的核心概念是实体和关系。实体用于描述问题域中的事物，它包括一组描述事物数据信息的属性；关系描述问题域实体之间在数据方面的联系。

信息建模方法早期的处理策略是：首先列出问题域中认识到的属性，然后将属性放到实体中，并添加实体之间的关联，用关系表示。在必要的情况下，运用父类型/子类型来提取实体属性的共性。

信息建模经过发展和改进之后也逐渐向面向对象靠拢。发展之后实体也称作对象，从属性描述对象，其他部分基本相同。改进后的信息建模方法认识问题域的出发点是问题域中的具体事物，用模型中的实体对应，然后再找出实体中与问题域有关的数据属性，最后添加实体之间的关系。用信息建模方法建立的系统模型对问题域的映射比功能分解方法和数据流分解方法要好。但是这种映射还不完善，因为它仅建立了问题域中事物数据方面的映射关系，事物的行为特征并没有反映。所以该方法是对问题域的半直接映射。

与面向对象方法相比，信息建模方法有以下差别：

- (1) 强调的重点是信息建模和状态建模，不是对象建模；
- (2) 没有把对于实体属性的操作（动态特征）封装到实体中，所以对象只有属性而没有服务；
- (3) 父类和子类之间只体现属性继承，不支持服务继承；
- (4) 没有采用消息通信。

### 6.2.3 从认识论看面向对象

面向对象技术是在 20 世纪 80 年代面向对象的程序设计语言的基础上发展而来的。同时，也是人们的认识不断提高的结果。所谓认识，是指在系统所要处理的问题范围（问题域）以内，通过人的思维对该问题域客观存在的事物，以及对所要解决的问题产生正确的认识和理解，包括弄清事物的属性和行为及彼此之间的关系，找出解决问题的办法。在认识问题域的同时需要某种方式将问题域的特征及解决办法描述出来。

从认识的角度来看信息系统分析和设计的过程：它是综合了三个方面来理解、描述和实现信息系统的各项功能的，如图 6.5 所示。

(1) 认识角度：人们认识问题的基本思维方法是归纳和演绎，它们都是抽象。在运用归纳和演绎的抽象原则的过程中，从不同角度看待同一个问题得出的结果也不完全相同。关于对信息系统的认识，历史上人们曾尝试从不同的角度来描述信息系统。例如，从过程的角度出发产生了面向过程的方法；从数据的角度出发产生了面向数据

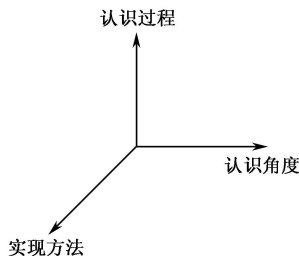


图 6.5 信息系统认识的三个方面

的方法；从功能的角度出发产生了面向功能的方法；从数据流动的角度出发产生了面向数据流的方法；从对象的角度出发产生了面向对象的方法；等等。这些方法反映了人们为了更好地认识事物所作的努力和尝试，同时也说明了信息系统本身的复杂性，因此，单从任何一个角度对信息系统的描述和认识都具有片面性，必须从信息系统的组成的本质方面认识和综合描述才能从根本上对信息系统有一个正确的理解和描述。

(2) 认识过程：就是从“不知”到“知”的过程。在这个过程中，需要科学的方式和方法才能对问题域有正确的认识。事实上，这是一个循序渐进的过程。一般而言，首先需要从全局的角度对问题域有一个概括的了解，形成对问题域的组成、功能等方面总体概括的认识；然后，在此基础上进一步深入分析直到完全掌握问题域中各个部分的本质和特征，产生对问题域在不同层次和各个方面上的深刻、清晰的理解；之后，在深入分析的基础上运用从不同角度看待问题所产生的方法（如面向功能、面向数据、面向对象等）对问题域建模，得到正确反映问题域的模型；最后，在建立的模型的基础上具体实现这个模型。这个过程简而言之就是分析—设计—实现，可能需要多次反复才能得到正确的结果。由于认识角度的不同，所以认识过程所形成的模型是对问题域的一种反映。

(3) 实现方法：即采用什么样的手段，在问题域模型的基础上具体实现模型。在信息系统中，实现意味着按照分析和设计形成的对问题域的描述采用计算机手段具体实现设计模型。它主要包括计算机硬件和软件，其中程序设计语言起到了重要作用，不同的语言对信息系统产生的影响会波及分析和设计工作，例如，采用结构化语言和面向对象语言要求分析和设计时采用不同的方法。

综上所述，只有上述三个方面的有机结合才能构造满足实际需要的信息系统。然而，从历史发展过程来看，不同的认识角度、认识过程和实现方法的目的在于解决两个鸿沟，即语言的鸿沟、分析和设计的鸿沟。

## 1. 语言的鸿沟

开发人员对问题域的认识是人类的一种思维活动。人类的任何思维活动都是借助于他们所熟悉的某种自然语言进行的。而信息系统的实现最终必须采用一种计算机能够理解的语言来对系统进行描述，这种语言就是程序设计语言。

人们习惯使用的自然语言和计算机能够理解、执行的程序设计语言之间存在着很大的差距。这种差距被称为“语言的鸿沟”，如图 6.6 所示。语言的鸿沟也就是认识和描述之间的鸿沟。一方面，人借助自然语言所产生的对问题域的认识和描述远远不能被机器所理解和执行；另一方面，机器能够理解和执行的语言又很不符合人的思维方式。开发人员要跨过这条鸿沟，从思维语言过渡到程序语言。这种过渡没有一种准确、可靠的方法。

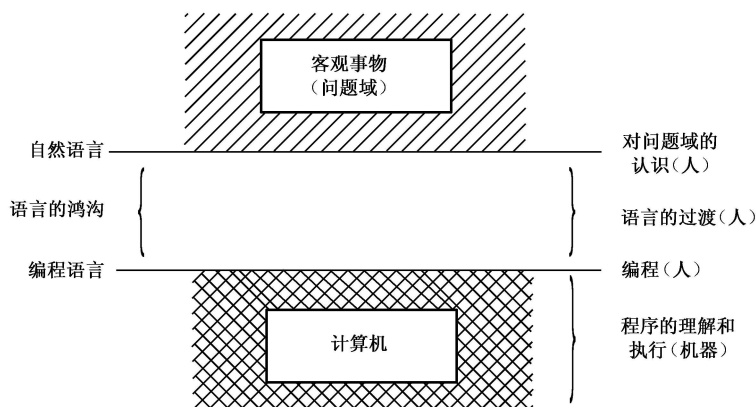


图 6.6 语言的鸿沟

随着程序设计语言由低级向高级的发展，语言的鸿沟在逐渐变窄。在计算机出现初期只有机器语言，整个语言只包括两种符号，即“0”和“1”，程序、数据等所有的描述完全由这两个符号构成。这种语言离机器最近，离人类的思维最远，毫无形象可言。这一阶段是语言鸿沟最宽的时期，其编程效率极低，极易出错。之后出现了汇编语言，它用比较容易理解和记忆的符号标识指令、数据等来构成程序。它比机器语言提高了一步，稍稍适合人类的形象思维，但两者的差距依然很远。这是因为，汇编语言的抽象层次太低，需要程序员考虑大量的机器细节。高级语言的出现是计算机程序语言的一大进步。它屏蔽了机器细节，提高了抽象层次，程序中可以采用具有一定含义的数据命名和容易理解的执行语句（如 Basic、Cobol）。20 世纪 60 年代末出现的结构化语言进一步提高了语言的层次。结构化的数据、结构化的程序等都使得程序

语言更加与人类的自然语言接近,但是两者之间仍然有很大的差距。面向对象的程序设计语言与以往各种语言的根本不同在于它能够更直接地描述问题域中的客观存在的事物及它们之间的关系,其封装机制、消息通信机制、继承机制和多态机制等更加接近对客观事物的反映。语言的鸿沟随着计算机程序设计语言的发展在逐渐缩小(与自然语言的差距仍然很大),如图6.7所示。

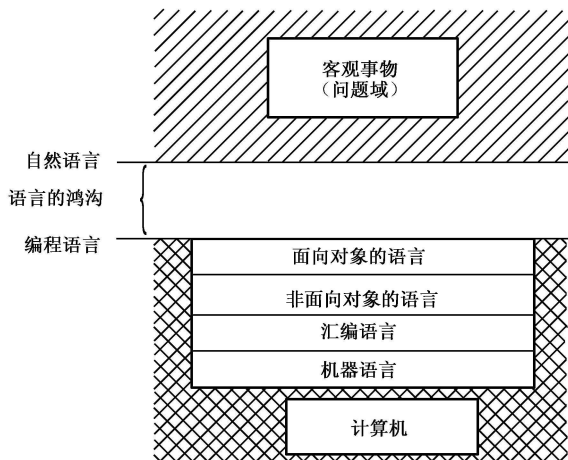


图 6.7 语言的鸿沟变窄

## 2. 分析和设计的鸿沟

早期的信息系统开发所面向的问题域相对简单,从认识清楚要解决的问题,到把这种认识用程序表达出来,对于能力较强的程序员还是不太难的一件事。然而,应用领域的扩大和问题域复杂程度的急剧膨胀,使信息系统的规模和复杂程度空前扩大。无论对问题域的理解和认识,还是用程序设计语言描述,都已经不是早期的程序可以比拟的,信息系统的复杂程度已经达到了开发人员无法控制的程度。这时就需要系统地、科学地从需求、分析、设计、实现等一系列的过程中提供的有效的方法来辅助人们对问题域的认识和理解,达到控制复杂问题的目的,从而开发出稳定、高效、满足需求的信息系统。

构建信息系统就是以计算机为手段来处理信息,提高信息处理的自动化程度。建设时,需要经过需求分析、系统设计、系统实现、系统测试等环节。分析和设计的鸿沟是指从问题域本身过渡到用计算机来反映问题域之间采用的认识方法、环节上的鸿沟,也就是采用什么样的方法填平问题域和计算机之间的鸿沟,使得从问题域到计算

机之间能够平稳过渡，计算机能够更好地反映问题域。结构化方法采用自顶向下、逐步求精的分析、设计方法和自底向上、逐步实施的实现方法，以数据流和构造模块的方式来认识和解决问题，无疑这种方法能够控制问题的复杂程度，完成信息系统的建设。它在历史上的确起到了重要的作用。然而，结构化方法对问题域的认识和描述不是以问题域中固有的事物为基本单位，并保持它们的原貌的，而是打破各项事物之间的界限，在全局范围内以功能、数据或数据流为中心来进行分析。例如，功能分解把整个问题域看作一些功能和子功能的组合，数据流法则把它们看作是数据在系统中流动和对数据的处理等。这些方法的分析结果不是直接地反映问题域，而是经过了不同程度的转化和重新组合，得到的结果或多或少与问题域有一定的偏差。另外，在系统分析、设计及实现时完全可能采用不同的表示方法，如分析时采用数据流方法，设计时采用模块结构图方法，实现时按照功能方式实现。由于采用不同的方法描述问题域均不是问题域的直接映射，所以这样经过几次表示的变化得到的结果与问题域本身之间的差距将越来越大，而且错误出现的可能性也越大，最终使得系统无疾而终。

面向对象的方法提供了更为有效的方法来构造信息系统。它既提供了从一般到特殊的演绎方法手段（如继承等），又提供了从特殊到一般的归纳形式（如类等），一般包括面向对象的分析、面向对象的设计、面向对象的实现、面向对象的测试和维护。面向对象的方法与传统方法的对比如图 6.8 所示。

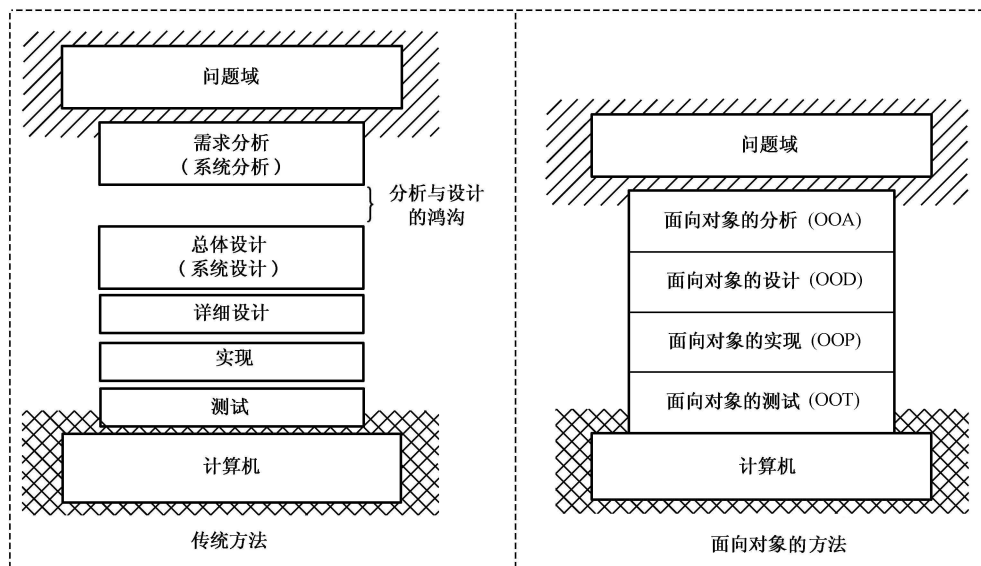


图 6.8 面向对象的方法与传统方法

面向对象的分析强调针对问题域中客观存在的事物设立分析模型中的对象，用对象的属性和行为分别描述事物的静态和动态的特征、行为，强调属性和行为与客观事物一致；用类描述具有相同属性和行为的对象群；用对象的结构描述客观事物的分类和组合特征；用消息连接、实例连接表示事物之间的动态和静态联系。可以看出，无论问题域中的单个事物，还是各个事物之间的关系，分析模型都能够保留它们的原貌，没有加以扭曲和转换，也没有打破原来的界限而重新组合，因此面向对象的分析模型能够很好地映射问题域。

面向对象的设计建立在面向对象分析模型的基础上，从设计实现的角度考虑人机界面、数据存储、任务管理等因素，仅在局部进行调整、修改，不存在表示方法上的变化，因此其设计结果仍然是问题域很好的反映。同样，面向对象的实现和面向对象的测试、维护等也同样没有改变对问题域的认识和表示。

综上所述，面向对象方法不但缩小了语言的鸿沟，而且基本填平了分析和设计的鸿沟。这种方法为信息系统的建造提供了一条全新的途径，而且经过时间和实践检验，面向对象方法的确在许多方面优于传统的分析和设计方法。

从本质上讲，面向对象的产生是人们的认识向自然的回归。在面向对象出现以前，曾出现了诸如面向功能、面向代数、面向过程、面向数据、面向应用、面向数据流等许多的方法，这些方法尝试从不同的角度和思路来认识客观事物，并试图能更加全面、深入、正确地反映客观事物。尽管这些方法在历史上的确起到了重要的作用，他们的努力也令人敬佩，同时也为面向对象的产生提供了很好的借鉴作用，然而，这些方法没有直接而全面地反映问题的本质。经过多年的努力和无数曲折的道路，人们认识到基于计算机的信息系统就是要对所处理的问题域有正确的认识，并把这种认识正确地描述出来。既然如此，就应该直接面向问题域中的客观事物来进行信息系统建模，这就是面向对象。实际上，人类认识世界中形成的普遍有效的方法就是在面向对象中的直接归纳和体现。在信息系统开发过程中也同样可以运用上述方法，不仅如此，对于人们在日常生活中习惯的思维和表达方式，也应当尽可能地采用，这就是面向对象方法所强调的基本原则。可以说，面向对象方法的观点与人们认识客观世界的方法在本质上是一致的，是认识向自然的回归。

#### 6.2.4 面向对象的几种方法

自从面向对象为人们广泛接受后，在信息系统开发中出现了许多基于面向对象的分析、设计方法。其中具有代表性的方法有 Coad-Yourdon 的 OOA 和 OOD 方法、Rumbaugh 等提出的 OMT 方法、I. Jacobson 等提出的 OOSE 方法、Wirfs-Brock 方法、



RDD 方法、Booch 方法及 IBM 公司的 VMT 方法等。

上述各种不同的方法，其出发点都是基于面向对象的，只不过各种方法在运用面向对象时各有侧重。在实际运用面向对象的方法时，不同组织在建设信息系统的过程中，一般根据组织自身的特点，综合不同的面向对象方法建立自己的面向对象标准，同时还结合一些传统的、成熟的分析与设计方法，最终形成满足自身需要的信息系统的建设规范。

下面对一些具有代表性的面向对象方法作概括介绍，本书将在后续章节中重点介绍 Coad-Yourdon 的 OOA 和 OOD 方法。

### 1. OMT (Object Modeling Technique) 方法

OMT (Object Modeling Technique) 方法覆盖了应用开发的全过程，包括分析、设计和实现。在分析阶段，OMT 方法强调对系统和相关领域的理解，通过分析，确立对象、关系、事件流和功能，并在此基础上建立模型。

分析阶段的模型由三部分组成：对象模型、动态模型和功能模型。对象模型表示了系统静态的、结构化的特征，通过图示的方式描述系统中的对象和对象间的关系，以及对象的属性和操作。动态模型表示了瞬时的、行为化的系统的“控制”性质，反映系统的动态特征，它包括事件图 and 状态图。OMT 方法使用脚本 (Scenario) 描述对象间的相互作用，事件图描述参与某个脚本的对象和事件，状态图描述系统中对象的各种状态及触发它们之间相互转换的事件。功能模型使用数据流图描述对象操作的具体含义。

OMT 方法的设计阶段由两个部分构成：系统设计和对象设计。系统设计负责划分子系统，确定系统的体系结构。对象设计的主要任务是细化分析阶段得到的模型，实现问题领域到计算机领域的转换。实现阶段的细节与具体的实现环境有关。

OMT 方法突出的特点是在分析阶段，它可以较为全面地描述系统的静态结构，因此 OMT 方法适合于数据密集型信息系统的开发。

### 2. OOSE (Object-Oriented Software Engineering) 方法

OOSE (Object-Oriented Software Engineering) 方法是一个使用事例 (Use Case) 驱动的方法，它建立的所有模型都是以使用事例模型为基础的。如图 6.9 所示，使用事例就是“用户和系统在一次相互作用中相关事务的一个特殊序列”。OOSE 方法的主要特点是能够较好地描述系统的需求。该方法适合于商务处理方面的应用开发。

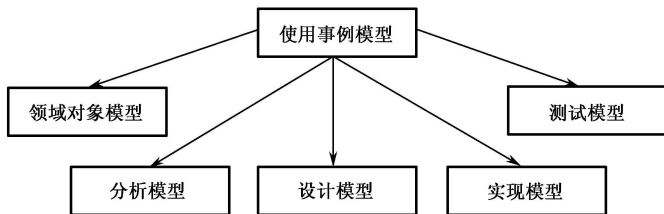


图 6.9 OOSE 方法

### 3. RDD (Responsibility Driven Design) 方法

RDD (Responsibility Driven Design) 方法（由 Wirfs Brock 于 1990 年提出）强调对象行为及对象间的关系，它用拟人的手法把对象的行为比喻为责任，对象间的关系比喻为合作，具有责任的对象比喻为合作者。RDD 方法认为合作者在系统中的角色不是顾客就是服务者。顾客向服务者请求某项服务，服务者则根据不同的请求提供相应的服务。RDD 方法把所有请求按照所请求服务的种类分类，组合成为合约 (Contract)，每个合约都详细说明了相关服务的每一步操作。

RDD 方法把开发过程分为两个阶段：探查 (Exploratory) 阶段和分析阶段。探查阶段的主要任务是寻找类，确定责任和合作。分析阶段的工作主要是细化对象的行为和服务，具体包括定义类接口、实现操作规范、定义类的多态性和细化服务规范。RDD 方法使用了 CRC (Class Responsibility Collaboration) 卡片来确定类的责任和它们的合作者。该方法对于小型项目具有较好的灵活性和适应性。但是对于大型系统，这个方法则显得力不从心了。

### 4. Booch 方法

Booch 方法把系统的开发工作分为两个过程：微观过程和宏观过程。

微观过程主要用于建立一个反复的、递增的开发框架，而宏观过程用来对微观过程进行控制。Booch 方法的微观过程是由脚本和产品的体系结构驱动的，它由四步组成，如图 6.10 所示。

Booch 方法的宏观过程控制开发过程中的许多活动，这些活动有益于开发人员评估开发风险并且及时纠正微观过程中的各种错误。宏观过程关心的是开发过程中的管理方面。宏观过程由五步组成，如图 6.11 所示。Booch 方法将工作集中在开发过程中的设计阶段，该方法对于开发的各阶段没有明确的划分，并且该方法没有规范需求说明书阶段。



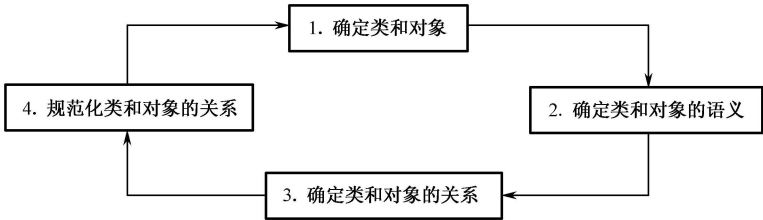


图 6.10 Booch 方法的微观过程

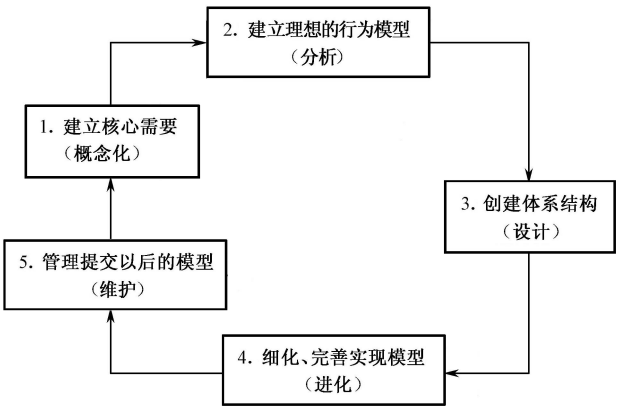


图 6.11 Booch 方法的宏观过程

进入 20 世纪 90 年代中期，各种面向对象的开发方法开始取长补短，相互融合，出现了一些新的方法。另外，原有的一些面向对象方法也进行了修改或更新。这些方法，无论是原有的，还是新生的，都力求在表示手段、代表语义和开发过程上都有所改进。其中具有代表性的是 VMT 方法。

5. VMT（Visual Modeling Technique）方法

VMT（Visual Modeling Technique）方法是 IBM 公司于 1996 年公布的方法。作为第三代面向对象开发方法，VMT 方法结合了 OMT、OOSE、RDD 等方法的优点，并且结合了可视化编程和原型技术。VMT 方法选择 OMT 方法作为整个方法的框架，并且在表示上也采用了 OMT 方法的表示。但是由于 OMT 方法中功能模型的描述采用数据流图，这与 VMT 的整体方法有些脱节，因此在 VMT 中摒弃了这部分内容，取而代之的是 RDD 方法中的 CRC 卡片，用它来定义各个对象的责任（操作）及对象间的合作（关系）。此外，为了加强需求分析，VMT 方法引入了 OOSE 方法中的使用事例

的概念来描述用户与系统之间的相互作用，确定系统为用户提供的服务，从而可以得到准确的用户需求。

VMT 方法的开发过程分为三个阶段，即分析、设计和实现。分析阶段的主要任务是建立分析模型。分析模型中包括使用事例模型、分析阶段原型、对象模型、动态模型和 CRC 卡片。它们之间的关系如图 6.12 所示。

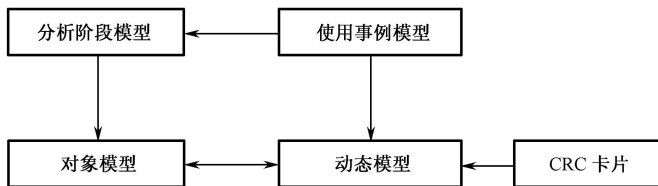


图 6.12 VMT 方法的分析模型

VMT 方法的设计阶段由三部分组成：系统设计、对象设计和永久性对象设计。所谓永久性对象，就是对象的生命期超过程序的执行期而长期存在。系统设计的主要工作是划分子系统，设计系统的体系结构。对象设计的主要工作是从设计的角度细化分析阶段得到的对象模型、动态模型和 CRC 卡片。永久性对象设计主要考虑与数据库相关的设计，如果系统开发不涉及数据库，则这一部分可以忽略。

VMT 方法实现阶段的主要工作是在某一种环境中实现系统。IBM 公司在推出 VMT 方法的同时也推出了支持 VMT 方法的可视化编程工具 VisualAge，通过它可以方便地实现设计阶段建立的模型。

## 6.3 面向对象的分析

人类认识世界的过程从本质上讲就是对事物进行抽象和区分的过程，典型的过程包含三个方面的内容。

(1) 从现实世界中区分特定的客体及其属性。例如，以一辆汽车及其大小和形状，来区别于其他事物。

(2) 区分事物的整体及其组成部分。例如，区分一辆汽车和组成这辆汽车的车轮、车门和方向盘等部件。

(3) 对不同种类的事物给出形式化的表示，然后在此基础上加以区分。例如，给出汽车和飞机这两类事物的形式表示，以区分这两类对象。

面向对象分析的概念和方法，就是建立在以上三种思维组织模式之上的。它反映了面向对象方法的客观性和自然性。

### 6.3.1 面向对象的分析原则

面向对象的分析方法是采用面向对象的思想形成的分析方法。它必须符合一些控制复杂性的原则及面向对象的基本思想，从而对问题域产生正确的、恰当的和简明扼要的认识，下面所提到的原则在分析过程中应尽量应用，这对于提高分析质量和效率是非常重要的。

#### （1）抽象

认识事物本身就是舍去事物非本质的、非共同的特征，将共性和本质的特征抽取出来的过程。在面向对象的分析中大量运用了抽象的原则。它将问题域中的所有事物的抽象看作对象，从对象的角度再进一步分析。在抽象的过程中，将问题域中的各个具体的个体存在的差异舍去，得到一个与问题域中的事物一致的抽象对象，即事物的直接映射。例如，对于企业中的“设备”，各个具体设备总是存在差异的（如具体的设备编号、型号等），但是“设备”对象中抽象的是所有设备所具有的共同的特征。另一方面，抽象需要“适度”，即问题域中的事物是复杂的，我们并不需要描述事物的一切，只需要研究与系统目标相适应的事物的本质特征就可以了，对于与系统目标无关的特征，即使是共同具有的特征也应该舍去。例如，在企业固定资产管理中，设备是其中的重要组成部分，抽象时应将设备的来源、型号、数量、价格及使用情况等特征抽取出来形成对象的特征，而对于设备的性能指标、工艺对象等方面则可以舍去。

对象中的属性（数据）和操作（功能）是其重要的两个部分。其中属性需要运用数据抽象，即需要定义对象自身的数据项，这些数据项必须是对象自身拥有的，只有对象才能够对这些数据项进行处理。在抽象过程中，需要明确有哪些数据是对象必须拥有的，哪些是必须舍弃的，同时需要定义这些数据项的数据类型和取值范围；操作需要运用过程抽象，即通过功能分解的方法完成对象的功能抽象，分析时可以在不同的抽象层次上考虑问题，而不必考虑功能是如何实现的，只须考虑对象应具备的功能即可。

#### （2）封装

封装可分为两个方面，从对象外部来看，封装将对象内部的细节掩盖起来，仅保留对外可见的接口，体现了对象的独立性，符合信息隐蔽的原则；从对象内部来看，封装将对象的属性和方法结合成为一个不可分割的整体，体现了对象的整体性。

#### （3）继承

特殊类（子类）对象拥有其一般类（父类）的全部属性和方法，即特殊类对一般类的继承。一般类的抽象程度高，往往代表某一类事物的全部，而特殊类的抽象程度

较低，作为一般类的子类，除具有一般类的所有属性和方法以外，还具有其他自身的属性和方法。

在面向对象的分析中，在一般类中定义的属性和方法，不必在特殊类中定义，继承本身表明特殊类中已经隐含地、自动地具备一般类的属性和方法。这样使得系统的对象模型比较简洁，也比较清晰。

#### (4) 分类

分类（Classification）就是把具有相同属性和方法的对象划分为一类，用类作为对象的抽象描述。在面向对象中，所有的对象都是通过类来描述的。对属于一个类的多个对象并不是单独描述的，而是通过“刻画”类的属性和方法来代表属于该类的所有对象。对象和类之间是“is-a”关系，特殊类和一般类之间是“is-a-kind-of”关系。

#### (5) 聚合

聚合（Aggregation）又称为组装（Composition）。其原则是，把一个复杂的事物看成是由若干个简单事物构成的，即整体和组成部分的关系，形成整体一部分的结构。部分对整体是“is-a-part-of”关系，反之是“has-a”关系。例如，汽车与发动机和底盘之间就是整体和部分的关系。

对过于复杂的对象描述，聚合从中分离一些独立的部分，从而简化了对整体的描述。对象的各个部分可以方便地增加和改变，这有益于随需求的变化而变化。

#### (6) 关联

关联（Association）是人类思考问题时常用的方法，即通过一个事物联想到另外一个事物，产生联想的原因是事物之间存在着某种固有的联系。在系统模型中，关联用来明确表示对象之间的静态关系。例如，学生和课程之间就存在一种关联。如果这种关联是需要的，则在 OOA（面向对象的分析）中可以明确地表示这种联系。

#### (7) 消息通信

对象和对象之间的信息交换方式只能通过消息进行，不存在对象之外的事物直接存取对象的属性。需要说明的是，在许多面向对象的语言中（如 C++），可以直接引用对象的属性，这样做是从语言本身和实现效率等方面考虑而采用的方法，不是面向对象方法本身。在面向对象的方法中，对象只有在获取外部的激励（消息），同时具有接收并处理消息的能力时，才表现出某种响应（功能）。如果对象没有接收某个消息的能力，则即便是激励到来，对象也无动于衷。“对牛弹琴”这个成语恰恰能够反映这一点。

#### (8) 粒度控制

人们在研究一个问题时，既需要宏观思考，也需要微观思考。对于一个复杂的问题，不可能在同一时间将宏观和微观都考虑清楚，这时就需要粒度控制。在分析问题

时，首先从宏观出发，找出其中的主要的组成部分，不考虑其中的细节，进而再考虑某个部分，而不考虑其他部分，这就是粒度控制原则。在 OOA 中，引入主题（Subject）的概念提供粒度的控制手段。

上面讲到的原则是面向对象分析过程中需要运用的主要原则。在分析时只有遵循这些原则进行分析工作，才能得到问题域的正确映射。

### 6.3.2 面向对象分析的模型和过程

#### 1. 面向对象的分析模型和组成

运用上面讲到的原则，按照面向对象的思想进行分析所得到的关于问题域的映射所形成的模型就是 OOA 模型。OOA 模型中的主要构成部分是类图（Class Diagram），其描述包括类、属性、方法、一般—特殊结构、整体—部分结构、实例连接和消息连接。其表达的信息可以分为三个层次。

（1）对象层：给出系统中所有反映问题域与系统目标相关的对象。用类的符号表达每一个类的对象。类作为对象的抽象描述，是构成系统的基本单位，也是面向对象方法观察问题域的基本单位。

（2）特征层：用于刻画类（代表的所有对象）的内部特征，需要描述每个类的属性和方法。它描述了对象的内部构成情况，给出了对象的内部细节描述。

（3）关系层：定义类之间的相互关系。这些关系包括：继承关系（分类关系），用一般—特殊结构表示；组装关系，用整体—部分结构表示；关联，反映对象的静态关系；消息，反映对象的动态依赖关系。这个层次描述了对对象外部之间的相互关系。

上述三个层次相互交织在一起，构成了 OOA 分析模型的基础。除此之外，还可以有主题图来控制粒度，也可以附加诸如用例图（Use-Case）、状态图和交互图等描述系统的其他特征。

#### 2. 面向对象的分析过程

面向对象的分析过程由以下主要活动组成。

（1）发现对象，定义问题域的类。

（2）识别对象的内部特征。

① 定义属性。

② 定义方法（服务）。

(3) 识别对象的外部关系。

① 建立一般—特殊结构。

② 建立整体—部分结构。

③ 建立实例连接。

④ 建立消息通信。

(上述三个活动建立基本模型)。

(4) 划分主题，建立主题图。

(5) 定义 Use-Case，建立交互图。

(上述两个过程在某些情况下可以省略)。

(6) 给出详细说明。

(7) 原型开发。

在进行 OOA 分析的过程中，上述活动及其子活动没有特定的次序要求，并且可以交互进行。运用时可以按照工作习惯采用某个次序或交替进行。

在具体运用时有如下建议，可供参考。

(1) 发现对象、定义属性/服务、建立结构/连接三个过程可以安排得比较近。可根据需要随时切换。例如，发现了问题域中的一些对象后，就可以着手定义它们的属性和方法，此时如果认识到某些结构，则及时建立结构和关系。

(2) 划分主题在很小的系统中可以省略；在中小规模的系统中，可以先建立基本模型后再根据对象的作用划分主题；在分析大型系统时，可首先在初步认识系统的基础上先划分主题，然后再根据主题进行分工，最后开始分析。

(3) 分析需求时可首先确定系统的边界和功能，可在分析工作开始时进行。交互和其他活动一般在其他部分基本完成后再对模型进行补充。

(4) 在原型开发活动中基本认识了系统（建立了一些对象）后，就可以建立一个最初的原型，随着分析工作的深入不断进行增量开发。

### 3. OOA 分析工作开始前的必要条件

在面向对象中，问题域中的所有事物都被看作是对象，它是构成系统的一个基本单位。标识对象就是要在问题域中找出与系统目标相关的对象。由于现实世界是纷繁复杂的，从不同的角度、对不同的人 and 不同的层次粒度出发对同一事物得出的映像都不完全相同，因此，标识对象工作在面向对象分析工作中具有重要的作用，只有那些找出与系统责任（它可以理解为最终构成的系统肩负着完成问题域中各项功能的使命，这是它的责任）相关的对象并将其抽象成为类，得到的最终系统才能反映问题域的本质。为此需要做以下工作。

### （1）研究用户需求，明确系统责任

分析人员在收到用户的需求和一些基本资料后，需要做的第一步工作就是首先要弄清楚用户的需求，得到一份正确的需求文档，知道用户到底想“做什么”。在需求尚没有明确、清晰以前，分析工作不宜立即展开。

分析人员可以采用“阅读”的方式查阅与用户需求相关的书面资料；与用户面对面交流，排查疑点，找出并明确用户不切实际或不正确的表达；也可以进行实地调查，彻底摸清实际情况，并将收集和记录的材料进行汇总、整理，得到一份清晰、明确的符合开发规范，明确系统责任的需求文档。上述方法对分析是必要的，这个过程需要确定将来的系统是从什么角度来看待问题域的，也为将来找出对象打下基础。

### （2）研究问题域

问题域（Problem Domain）就是被开发的应用系统所考虑的针对整个业务的范围。开发得到的系统就是问题域的一种映射。这个映射是否正确、合理，需要对问题域进行深入的研究后才能判断。

研究问题域的最可靠的方法就是身临其境，仔细观察，在现场直接获取问题域中存在的各种事物，包括不同职能的人员、使用的设备、组织机构、管理模式、工作流程、实物、单据、报表等。

研究问题域最终必须明确以下两点。

① 用户需求必须明确，这是最基本，也是必须细致去做的工作；

② 建立一个符合问题域情况、满足用户需求的分析模型。这个需求分析模型可以采用各种适用的方法完成，如采用功能分解法得到系统的功能情况，采用数据流分析法找出数据的流动和加工情况。问题域研究得越深入、越清晰，OOA 分析就越准确。

### （3）确定系统边界

在完成上述两项工作后，明确最终系统涉及的业务范围和外部世界的接口。在系统内部是系统本身包含的对象；在系统边界以外是系统外部的活动者（Actor），主要包括人、设备和外部系统三种。

## 6.3.3 标识发现对象

标识发现对象的根本出发点是问题域和系统责任。二者从不同角度告诉系统分析员应该设立哪些对象。从问题域出发，侧重于客观存在的事物与系统中对象的映射；从系统责任出发，侧重于系统责任范围内的职责由相应的对象完成。二者的范畴在很大程度上是重合的，但有区别。对分析员来说，只考虑问题域，不考虑系统责任，则不容易区别哪些对象是应保留的，哪些是应舍弃的。反之，如果只考虑系统责任，则



容易受其他分析方法的影响（如功能分析法），过多考虑系统内部对象之间的责任，而可能忽略对象所反映的真实问题域。

其次，正确运用抽象的原则，即分析应该把问题域中的事物映射为什么对象，以及如何对这些对象进行分类。例如，在图书馆管理系统中，需要建立“书”类，而且需要将每一本书作为一个对象来考虑，这是因为需要记录每一本书的借还情况；在书店管理系统中，没有必要将每一本书都作为一个对象来看待，只要将同一种书看作一个对象，记录它们的数量、出版社、书号及单价等信息就可以了。再如，对企业中的人员，在某种情况下我们需要区分哪些是管理人员，哪些是一般人员，而在另外的系统中，只须以“员工”来说明就足够了。上述两个例子说明，在不同的情况下，对客观事物的抽象程度、角度及层次是不完全相同的。系统分析人员需要根据用户的需要并结合问题域的特点，清晰、准确、简洁地描述问题域，所有这些必须恰如其分地运用抽象原则，在深入理解问题域的基础上，经过缜密思考和取舍而逐步达到。这是一个反复提高的过程。

标识发现对象的过程可以由五个方面得到，如下所述。

### 1. 寻找对象

如果不加任何限制，则在面向对象的世界里，所有事物皆是对象。显然，寻找对象的任务是分析人员在这些对象中找出问题域需要的对象，即寻找那些与系统责任相关的各种事物。在这一步，分析人员需要将对象看作是一个“黑箱”，抛开对象的内部的细节，也可以不去理会对象之间的关系，更无须考虑对象将如何实现，而直接从面对的问题出发，详细列出与问题域和系统责任相关的所有存在的对象。需要强调一点，这些对象可以是实实在在存在的，也可以是实际不存在的（抽象的对象），只要与问题域相关，即可先将它找出来，再结合系统责任对其进行区分和取舍。分析人员应该“先松后紧”，毕竟，完成取舍和筛选的工作要比将来发现遗漏后再增补来说，它的影响要小得多。

### 2. 哪些可以作为对象

为了发现各种可能的候选对象，需要从问题域、系统边界和系统责任三个方面考虑，找出候选对象。

#### （1）问题域

问题域中可以作为对象的因素一般包括人员、组织、物品、设备、事件、表格和结构等。

① 人员：一般系统中都会涉及人员。其中一类是系统中需要保存和管理的人员，



如人事管理系统中企业的员工；另一类是在系统中承担某种角色的人员，如销售经理在系统中的角色与一般操作员所扮演的角色就不同，至少在功能使用权限上二者是有区别的。

② 组织：在系统中发挥一定作用的组织机构，如工作班组、职能部门、派出机构等。

③ 物品：需要系统管理的各种实物，如企业的生产的设备、产品、生产材料、经营的商品等。还需要注意一些无形的事物，如企业生产计划、学校的课程等。

④ 设备：系统中动态运行并由系统进行监控或使用的设备，如仪器仪表、运输工具等。尽管也可以将它们列为物品一类，但是在系统中，它们具有动态特征，所以单列出来考虑。

⑤ 事件：是指那些需要系统长期记忆（存储）的事件。在一个系统中或多或少地都要发生一些事件，有些事件需要长期保留，而有些则不需要。例如，在财务管理系统中，资金支出事件必须记录在案，否则违反了财务规定，而查询当前资金情况的事件则不需要记录。再如，一般的软件在使用时（使用事件），不需要记录是谁在使用，而在银行管理系统中，不但需要记录使用者，而且还要记录使用了哪些功能及时间。从上面的例子可以看出，分析人员需要具体问题具体分析，以判断哪些是必须长期存储（记忆）的，再根据其复杂程度确定是否需要设置相应的对象。

⑥ 表格：问题域中的表格是多种多样的，如业务表格、统计表、登记表、成绩单、订单、支票、账册等。这些表格在分析时很容易获得，而且建立相应的表格对象似乎是自然而然的。虽然这种做法通常是正确的，但是不加分析地去做，有时并不十分合适。

⑦ 结构：在问题域中，事物之间存在结构关系；在面向对象中，需要确定对象的一般和特殊关系，以及整体和部分的关系。例如，在车辆管理中，不同类型的车辆之间存在着一般和特殊的关系（如卡车、客车都是车辆），底盘和发动机与车辆是部分和整体的关系。通过事物之间的结构可以发现相关的对象。

## （2）系统边界

从系统外部边界出发，寻找系统以外的与系统直接交互的相关人员、外部系统、设备，从而找出对象。其中，人员可以是操作使用人员、系统用户；外部系统是与系统有信息交换的系统，例如，需要从外部系统获取或发送相关数据，这时需要增加相关的对象以完成与其他系统的信息交换；设备是系统外部的活动者或设备与系统相连并交换信息，如企业自动化管理系统需要从生产线的设备中获取运行的状态、参数等信息，可以设置这些设备的对象以便完成设备的各种操作和控制。

## （3）系统责任

尽管通过上面两点已经能够发现系统中的许多对象，但是不能够明确表示这些对

象一定是完整的，没有遗漏的。系统责任则可以用来检查可能存在的疏漏。其方法是对照系统责任的每一项功能，查看这些功能是否有相应的对象来完成。如果没有任何对象能够提供此服务，则需要增加遗漏的对象。需要说明一点，对于诸如数据备份、恢复等功能一般在问题域中很难找到对应，可以考虑专门增加。另外，还有一些可能与具体的实现相关（如图形界面、数据库等）的，可以将它们推迟到设计阶段考虑。OOA 关注的主要是问题域本身。

### 3. 对候选对象的审查和筛选

在找到许多候选对象之后，需要对它们逐个进行审查，保留那些真正需要的对象，筛选、合并或精简那些不必要的对象。

#### (1) 舍弃无用对象

对每个候选对象，判断它们在系统中是否真正有用，可以审查对象的属性和方法看它们是否真正有用。对于属性，要仔细检查它们是否在系统中进行保存和管理，如果有则保留；对于方法，则要看该方法（服务）是否在系统中发挥作用。

在这个过程中，没有必要将对象的所有属性和方法全部找出来，不过在思考问题时，一旦发现就可以将它们填写到对象的符号表中，一旦找到这些属性和方法，就意味着对象在系统中是发挥作用的，需要保留。

#### (2) 对象精简

如果系统中对象的种类和数量过多，则将增加系统的复杂性，这时就需要考虑精简。这当中对于只有一个属性或者一个服务（方法）的对象应仔细考虑。例如，在教学管理中，“班主任”只有一个“姓名”属性，而这个对象被“班级”对象引用，此时可以完全把它合并到“班级”对象中，即增加一个“班主任姓名”的属性。

### 4. 质疑和调整

当分析人员对系统的对象有了正确、完整的认识后，建立对象的类（Class）相对就比较简单。通常对每一个对象定义一个类，用符号来表示。但是在下述情况下需要进行调整。

(1) 类的属性或服务不适合该类的全部对象：例如，“汽车”对象如果有“乘客数量”的属性，它只适合于客车，不适合于货车。此时需要进一步分类，考虑建立一般—特殊结构。

(2) 属性和服务相同的类：在问题域中对象反映的不是同一个事物，但是它们的属性和方法在系统责任上经过抽象后却可能完全相同。例如，书籍和计算器之间的差别明显，但是如果把它们放在商店销售管理系统中，则其属性和方法完全相同，可将

它们合并成为“商品”类。如果确实找不到一个合适的类来概括它们，则宁可不合并，以免造成概念上的混乱。

(3) 属性和服务相似的类：往往这些类之间或多或少都存在一些联系。对于类中属性和方法有许多相同的，则考虑建立继承结构；对于类中描述了相同的事物，则可以考虑建立整体一部分结构。例如，“汽车”和“飞机”对象中，都描述了“发动机”，则可以分离后建立“发动机”类。

(4) 同一事物的重复描述：问题域中某些事物实际上是另一种事物的附属物或一定意义上的抽象。例如，工作证对职员，车辆行驶证对车辆，图书索引卡对图书等。这些对象与它们所描述的原始对象之间存在信息冗余（重复描述），可以将它们合并，合并时增加相应的属性即可。

## 5. 如何命名对象类

类的名称应该能描述该类中所有对象的基本特征。命名类的主要方法有：

- 用单个名词或形容词加名词作为类名；
- 采用标准名称作为类名；
- 尽量采用可读性好的名字。

一般来说，一般类的名称采用反映群体的词汇，特殊类的名称采用反映个体的词汇。例如，“书籍”作为一般类名；“书”作为特殊类名；还有如“车辆”与“轿车”、“货车”、“客车”，也是这种命名法。

对象的认定是 OOA 的一件重要的工作，同时又依赖于具体的问题域和分析人员的经验，需要针对实际问题区别对待。

下面以商场销售管理系统为例建立 OOA 的模型。假定商场销售管理系统有如下功能需求：

- 为顾客选购的商品计价、收费、打印清单；
- 记录每一种商品的编号、单价及现有数量；
- 帮助供货员发现哪些商品将要脱销，以及时补充货源；
- 随时按上级系统的要求报告当前的款货数量、增减商品种类或修改商品定价；
- 交接班时结算货款数目，报告上级系统。

对这个例子而言，系统边界以外与系统进行交互的活动者有收款员、供货员和上级系统。据此，可以发现如下一些对象。

(1) 收款机：该对象直接与收款员这种活动者进行交互，模拟收款员的登录、售货和结算等行为。

(2) 供货员：此类对象用来与实际的供货员进行交互（提醒他们及时补充货物）并模拟他们的行为（在增加货物时修改系统中的商品数量），这些行为是从系统内部引发的，所以它是被动对象。

(3) 上级系统接口：用来处理与上级系统的交互。它的某些行为（如查账、更改商品的种类与价格）是由上级系统（而不是从本系统内部）引发的。

考虑该系统问题域内部的事物和系统责任，可以发现下述对象。

(1) 商品：这是该系统中最明显的对象。每一个对象实例表示一种商品，记录该商品的名称、价格、数量等信息，并通过相应的服务动态地保持这些信息的准确性。放在超市中销售的商品，一般都不需要单独地记录每一件商品的信息。所以，可把每一种（而不是把每一件）商品看作一个对象，把数量作为对象的一个属性（在其他场合，如买卖飞机、楼房、珠宝、名画之类的大型或昂贵的商品则另当别论）。

(2) 特价商品：这是一类较特殊的商品，该类商品在指定的时间内按销售价格（大多是优惠价格）销售，它有其特殊的属性。

(3) 计量商品：这是另一类较特殊的商品，其包装不标准，或者没有包装，需要在收款时按照它们的质量、长度或容积等单位进行计量，并按计量结果计算其价格。

(4) 商品一览表：考虑系统责任，为了在收款时能根据输入的商品编号快速找到相应商品的信息，需要设立一个“商品一览表”对象，它保持一个商品目录表，并提供对商品项的检索及增删等功能。

(5) 销售事件：顾客购买一组商品，只要是通过一次计价收款完成的，就称作一个销售事件。每个这样的事件都需要在系统中保存一段时间，以便汇成账目并在必要时复查。所以要设立“销售事件”对象。

(6) 账册：记录一个收款员在一个班次内经手的所有销售事件的款、货账目，负责向上级系统报账，并在换班时进行账目交接。它的一个对象实例只针对一个收款员的一个班次，而不是总账（总账在上级系统中）。

按常识，在一个商场中收款员和管理人员都应该是一些值得考虑的对象。假如已经把“收款员”和“经理”列为候选的对象类，现在考虑进行筛选，看看是否有必要保留这两个类。

本系统的功能需求没有要求对各类人员的信息进行计算机管理，所以各类人员对象是否有必要存在只是看这些人员的行为和个人信息是否对系统功能的履行起到一定的作用。

需求中没有包括对经理的工作进行计算机处理或提供辅助支持，经理本人的信息对于完成需求中规定的业务处理功能也没有用处。所以，不必设立“经理”对象。

收款员与系统的功能需求有密切关系，他们是与系统对话的活动者，所以系统应

提供相应的对象处理这种对话，在以上发现的对象中，“收款机”对象就是进行这种处理的。如果愿意，也可以把“收款机”对象改名为“收款员”，但是没有必要同时设立两类对象，因为根据系统的功能需求对实际的收款员或收款机进行抽象之后，所得的结果是相似的。如果功能需求中包括管理收款员的人事信息，则又另当别论。

系统中设立了“供货员”对象类，但并不是为了提供这类人员的信息，而是因为他们向店中补充货物的行为要在系统中有所反映。

通过这个例子可以看到，尽管问题域中有各种各样的事物，但是在以系统责任为目标进行抽象之后，有些从常识看来很值得注意的事却不需要设立相应的对象。系统中的任何对象都是为了提供某些信息或履行某些功能的。如果没有这些用途，则这种对象就没有必要在系统中存在。抽象的原则要求忽略与当前目标无关的事物特征，在OOA中当前目标就是系统责任。每个对象都忽略了实际事物的许多特征。如果某种事物的所有特征都与系统责任无关，则将被完全忽略，系统中不应出现描述这种事物的对象。

现在假设，这个商场销售管理系统除了本节开头列出的功能需求之外，还要求提供以下的功能：

- 提供商场中各类职员的信息；
- 经理通过系统为收款员安排班次；
- 经理在系统的帮助下发现何种商品畅销，何种商品滞销；
- 定期统计收款员完成的工作量并评价他们的工作效率。

如果增加了这样一些需求，那么系统的OOA模型，无论其对象层还是后面将建立的特征层和关系层，都将比现在的示例得到的结果复杂。读者可以根据这些新的需求对给出的OOA模型进行扩充，以此作为练习。

通过以上分析，建立的OOA模型的对象层如图6.13所示。

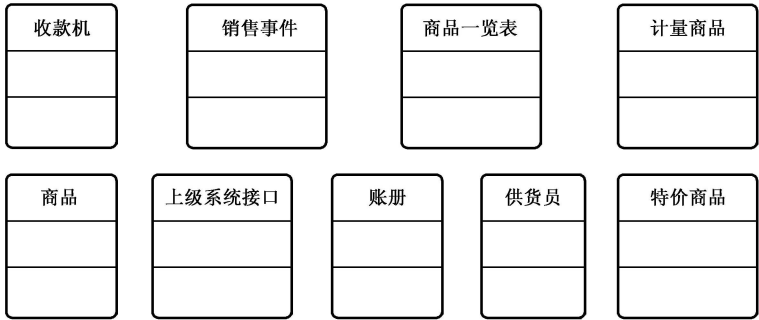


图 6.13 商场销售管理系统对象认定

### 6.3.4 定义属性

在确定系统责任的对象以后就可以定义属性。在 OOA 中，类是问题域中同类对象的抽象，定义属性就是寻找属于类的属性，这些属性是类所包含对象的所有个体中都应该具有的。

在分析过程之中，对属性的定义可以通过以下步骤进行。

#### 1. 认定属性

认定属性有以下几个基本原则。

(1) 该属性对该类中的所有对象都是普遍适用的，即属于类的所有对象（实例）都具有这个属性。例如，对于“机动车”类中的“发动机”属性，显然机动车中必然有发动机，因此这个属性是合适的，但如果定义了“柴油发动机”属性就不合适了，因为有许多机动车的发动机并不是用柴油的。

(2) 一个属性即使对某种对象的实例都是适用的，也还要考察在现实世界中它与这种事物的关系是不是最密切。例如，在一个车辆注册系统中考察“颜色”这个属性，对车辆进行注册的时候，都将产生一个注册车辆的事件，采用填写注册表（卡）的方式完成，这样“颜色”属性可以是注册表对象中的一个属性。表面看来这种做法与实际相符合，但是从关系的密切程度来看，“颜色”是机动车对象的固有属性，它们的密切程度比注册表高，因此，“颜色”属性应当认定在机动车对象中，而注册表仅仅在使用时用到了机动车的颜色属性。

(3) 认定的属性应当具有“原子性”，也就是说，属性不应再进行分割，而应当作为一个整体看待。例如，“人员”对象中的“住址”属性，它又包括国家、省、市、街道等内容，作为属性时不要将它们拆开后分别作为“人员”对象的属性，而应将“地址”看作一个整体的属性。

(4) 这个属性是否可以通过继承获得。一般类中已经定义的属性，特殊类中就不能重复定义。通过检查一般类中的属性可以消除重复定义。

(5) 一个属性是从其他属性直接导出的属性。如果一个属性明显是从另一个属性直接导出的，则应该去掉。例如，“年龄”和“出生日期”二者，就应该将“年龄”去掉。如果一个属性用其他属性经过复杂计算得到，则可以在 OOA 阶段暂时不做简化，保持模型的直观性，设计阶段再根据具体情况做相应的调整。

#### 2. 确定属性的位置

在对象的继承结构中，底层的对象拥有父对象的属性。哪些属性应该在父对象中？



哪些属性应该在子对象中？其基本原则是：只要某个属性在底层对象中都是共同具有的，就应该将它定义在上层对象中。

### 3. 属性规格说明

在属性规格说明中，需要将属性的意义和作用、数据类型、属性体现的关系及有关实现的要求（取值范围、初始值、度量单位、完整性、安全性等）给出详细的说明，包括属性的名称、描述、约束和范畴。

图 6.13 已经给出商场销售管理系统所包含的对象类，现在确定每个类的属性。

“收款机”对象的属性应表明当前是哪个收款员在本台收款机上工作，以及她（或他）本次工作的开始与结束时间。

“商品”对象的属性包括该种商品的编号、名称、单价、架上的数量及下限。下限是指当架上商品数量低于这个限度时，就及时提醒供货员补充货物。

“特价商品”类继承了“商品”的所有属性与服务，同时具有自己的特殊属性，应指明实行特价的时间范围。

“计量商品”类继承了“商品”的所有属性与服务。但“单价”属性的语义发生了变化，指的是一个计量单位的价格。补充的特殊属性有“计量单位”和“计价方式”（散装商品的计价方式是单价乘以计量结果，成件商品的计价方式是按标签上已经标出的价格收款）。

“商品一览表”的属性是汇总店内所有商品的“商品目录”。它是一个数组，每一个元素包括一件商品的编号并指向“商品”对象。

“供货员”对象的属性是一个缺货登记表，每当某种商品的架上数量低于其下限时，就在这个登记表中记录下来，以便提醒供货员补充货物。

“销售事件”的属性有：“收款人”，记录由哪个收款员在哪台收款机上处理这个销售事件；“购物清单”，记录顾客选购的每件商品的编号、名称、数量及价格；“应收款”，累计所有被选购商品的价格总和；售出日期及时间等。

“账册”对象的属性记录一个收款员在一个班次中的每个销售事件，并累计其销售收入。实际上，用一些指针（或对象标识）指向每个销售事件，就可得到它们的明细信息，因此，可将“账册”与“销售事件”组织成整体一部分结构。

“上级系统接口”对象设立一个“账册目录”属性，记录店内所有正在使用和已经结算的账册的指引信息，以便及时地找到它们。

根据以上对属性的分析，得到该系统 OOA 模型的属性层，如图 6.14 所示。

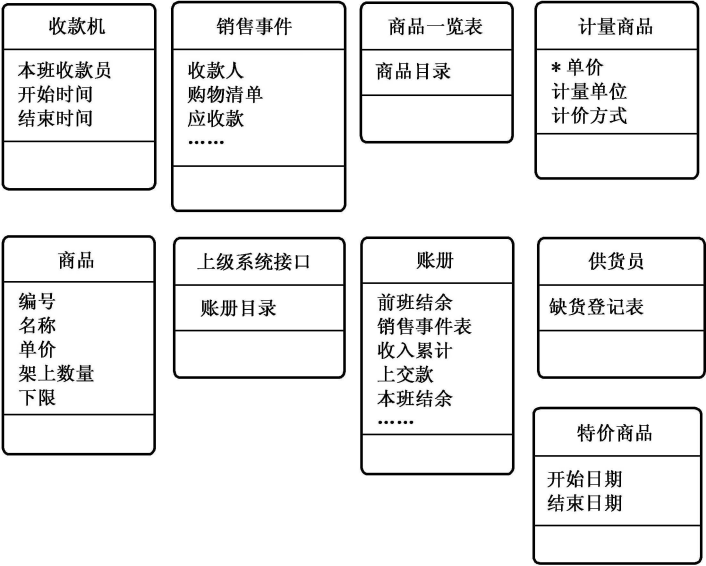


图 6.14  商场销售管理系统对象属性

6.3.5  定义方法

定义方法是指对象在接收到一条消息后所要完成的功能。方法是定义每个类及分类结构在系统责任约束下的行为。

1. 系统的行为

与对象有关的某些行为实际上不是对象自身的行为，而是系统把对象看作一个整体来处理时施加于对象的行为，如对象的创建、复制、存储、删除等。对于这类行为，除非特殊情况一般不需定义相关的方法，因为此类行为不是对象自身拥有，是系统施加于它们的，可以在设计阶段再考虑它们。

2. 认定对象方法

严格来讲，对象的属性操作仅仅允许对象自身完成，不允许外部对象访问，应由对象提供的服务（方法）完成。这样，要访问对象的一个属性就必须增加两个方法（读取、设置）。如果属性较多，则必然在对象中设置多个类似这样的方法，显然在分析阶段做并不合适。对于这些简单的方法在分析阶段可不必定义，留在设计阶段补充即可。



对于复杂的方法，则需要认真考虑。通常，从以下几个方面考虑方法。

(1) 从系统责任的角度发现方法。对象的服务直接地体现了系统的责任，含有实现用户需求的成分，因此从用户需求出发，提取每一项功能需求，找出功能应由哪个（哪些）对象承担，然后建立对象的方法。也可以逆向从对象存在的目的和能够为系统责任承担哪些功能的角度出发，找出方法。

(2) 从问题域中发现方法。对象在问题域中具有哪些行为？这些行为与系统责任的关系？设置哪些方法能够模拟问题域的情况？从而找出方法。

(3) 分析对象的状态。对象的可能存在的状态和状态的变迁过程，可以为发现方法提供线索。可以画出对象的状态转换图，然后，从状态的迁移中判断对象是否具有某种行为，从而找到相应的方法，也可以从引起状态转换的原因出发找出方法。

(4) 追踪服务的执行过程。

问题域中处理问题的过程和方法对发现方法有很大的帮助，通过模拟问题的处理过程，对于过程中相关的对象，看它们在解决问题过程中的责任如何，就可以找到对应的方法。

### 3. 对方法进行规格说明

对方法进行规格说明的重点是那些要求使外部可观察到的行为，这些行为作为将来对系统需求进行验证、对系统实现进行验收和测试的基准。

上面已经描述了商场销售管理系统所包含的对象类及对象的属性，现在分析每个类的方法。

“收款机”对象的方法包括：“登录”，使本班收款员开始工作，它是一个主动服务；“售货”，循环地为每个顾客计价收款；“结账”，在收款员下班或交班时结算本班的账目。

“商品”对象的方法有：“售出”，从架上数量减去已售出商品的数目，若剩余的数目低于下限则向“供货员”对象发消息；“补充”，当供货员补充了一些商品时，把补充数量与该种商品原先的架上数量相加；“价格更新”，由“上级系统接口”对象引用，修改本种商品的价格。

“特价商品”类继承了“商品”的所有属性与方法。

“计量商品”类继承“商品”的所有属性与方法。对于从“商品”类中继承的方法要重新定义，因为算法发生了变化，它们是多态的。

“商品一览表”的方法有：“检索”，通过编号查找相应的商品对象；“种类增删”，增添或删除“商品目录”中的商品项。

“供货员”对象的方法有：“缺货登记”，将告缺的商品名称及编号记到登记表中；

“供货”，在供货员补充了货物之后，向“商品”对象发消息以更改其数量，并删除缺货登记表中的相应的条目。

“销售事件”的方法有：“销售计价”，逐条记录商品清单，并累计应收款数；“入账”，将本次销售事件的信息计入账册。

“账册”对象的方法有：“接班”，记录从上一班收款员那里接收了多少未上交的货款（因为收款机上总需要保留一些零钱）；“记账”，记录每一个“销售事件”对象，并累计其收入金额；“报账交班”，向上级系统报账，记录上交的款数和移交给下一班收款员的款数。因此，这个对象还应该增设如下三个属性：“前班结余”、“本班结余”和“上交款”。此外还应记录该账册开始使用和结账的日期与时间等，这里不一一列举。

“上级系统接口”对象的方法有：“消息收发”，负责与上级系统通信，并通过引用其他方法完成上级系统要求处理的事项。由于这个方法要随时关注上级系统的请求，实现时应该是一个与本系统其他任务并发执行的进程，所以它是一个主动方法。其他的方法有：“查账”，按上级系统的要求查阅账目并报告结果；“报账”，从本系统主动向上级系统报告账目；“价格更新”，按照从上级系统传来的信息，更新指定商品的单价；“种类增删”，按照上级系统的信息增添或删除商品对象及其在商品一览表中的条目。

根据以上对方法的分析，得到该系统 OOA 模型的方法层的描述，如图 6.15 所示。

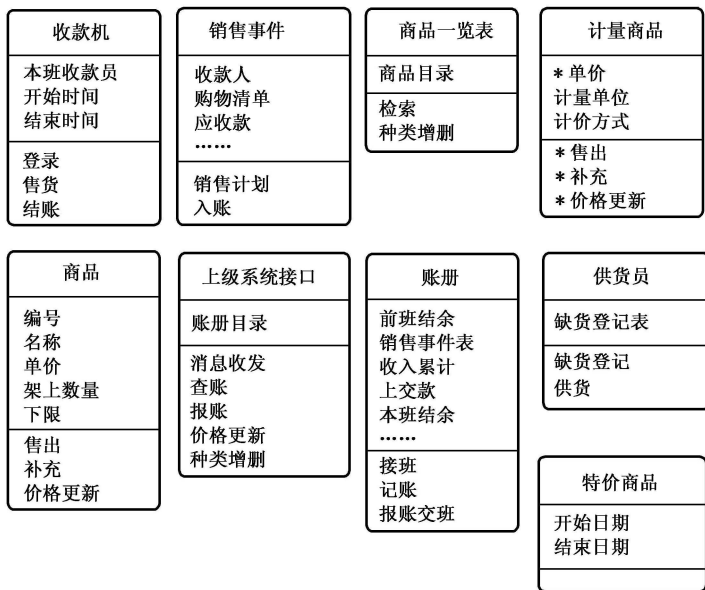


图 6.15 商场销售管理系统对象方法

### 6.3.6 标识结构和连接

在面向对象的分析方法中，结构指的是多种对象的组织方式，用来反映问题域中的复杂事物和复杂关系。这里，结构包含两种：分类结构和组装结构。关系包括对象之间的静态关系和动态关系。

分类结构针对的是事物的类别之间的组织关系，组装结构则对应于事物的整体与部件之间的组合关系。使用分类结构，可以按照事物的类别对问题域进行层次划分，体现现实世界中事物的一般性与特殊性。例如，在交通工具、汽车、飞机、轮船这几种事物中，相对一般的是交通工具，其他则是相对特殊的。因此，可以将汽车、飞机、轮船这几种事物的共有特征概括在“交通工具”中，也就是把对应于这些共有特征的属性和方法放在“交通工具”这种对象之中，而其他需要表示的属性和方法按其特殊性分别放在“汽车”、“飞机”和“轮船”这几个对象之中。在结构上，则按这种一般与特殊的关系，将这几种对象划分在两个层次中，如图 6.16 所示。

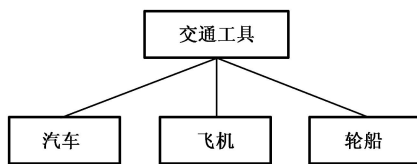


图 6.16 “交通工具”的层次

组装结构表示客观事物中整体与部件的关系。例如，如果把汽车看成一个整体，那么发动机、变速箱、刹车装置等都是汽车的部件，相对于整体就分别是一个局部。

静态关系是通过对象属性反映对象之间的联系，是对象之间固有的联系。如“学生”对象与“课程”对象之间存在选修关系。静态关系用实例连接表示。

动态关系是对象行为之间的依赖关系，它们通过消息方式关联起来而相互作用。动态关系通过消息连接表示。

#### 1. 分类结构

分类结构又称一般—特殊结构，是由一组具有一般—特殊关系（继承关系）的类所组成的结构。从类的特征出发可以定义一般类和特殊类。

如果类 A 具有类 B 的全部属性和服务，而且具有自己特有的某些属性或服务，则类 A 叫做类 B 的特殊类，类 B 叫做类 A 的一般类。

从集合论的观点看，特殊类对象实例集合是一般类对象实例集合的真子集；而一

般类的特征集合则是特殊类特征集合的真子集，如图 6.17 所示。

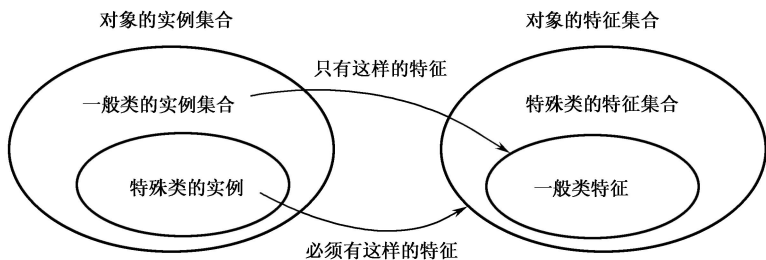


图 6.17 一般类与特殊类

一般类与特殊类之间的关系叫做一般—特殊关系，又称为继承关系或“is-a-kind-of”关系。这种关系是可传递的。如果一个特殊类只直接继承一个一般类，则这种继承称为单继承；如果一个特殊类直接地继承两个或两个以上的一般类，则称为多继承。现在给出分类结构的定义：

分类结构是把一组有一般—特殊结构关系的类组织在一起而得到的结构，它是一个以类为节点、以一般—特殊关系为边的连通有向图。

如果在分类结构中仅存在单继承，则它是一个以最上层一般类为根的树，称作层次结构；如果在结构中存在多继承，则它是一个半序的连通有向图，称作网格结构。

在 OOA 中，分类结构简化了类的定义，对象的共同特征仅在一般类中给出，特殊类通过继承自动拥有这些特征，从而不必再重复地加以定义。分类结构给出了类之间的继承关系，简化了特殊类的定义。

分类结构的表示法，是用一般—特殊结构连接符来连接该结构中的每一个类的，如图 6.18 所示。

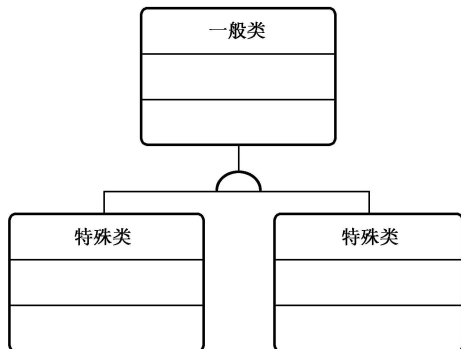


图 6.18 分类结构的表示法

认定分类结构,总的原则是先从一般向特殊考虑,再从特殊向一般考虑。

对于一种对象,首先认为它具有最一般的含义。这时,看它在问题域中具有不同特殊性的可能性,即对于该对象每一种可能的特殊性,考虑:

- ① 是否可以用不同的属性和(或)方法来描述;
- ② 是否反映了现实世界中有意义的特殊性;
- ③ 是否在问题域之内,即描述这种特殊性的属性和方法是否限定在目标系统之内。

在按上述原则确定了对象的特殊性之后,就可以将对象共有的属性和方法安排于一般含义的对象中,而将扩充的特殊属性和方法分属于特殊含义的对象。例如,当确定飞机具有民用和军用这样的特殊性之后,就可以把共有的属性和方法(如机型、航速、航程等)提取出来,让其从属于“飞机”对象;再把特殊属性和方法提取出来,如与民用飞机有关的班期、离港时间、到达时间等,以及与军用飞机有关的作战半径、作战用途、电子对抗装置等,然后引入“民用飞机”和“军用飞机”对象,令上述特殊性和方法分别属于这两种对象。

可以看出,从一般向特殊的方式考虑分类结构,可对问题域中出现的事物找出更为具体的内涵。

然后,再从特殊向一般考虑。对于一种对象,在认为它具有某种特殊的含义之后考察:

- ① 问题域中是否还有其他对象与这种对象有一些属性和方法是共有的,若有,则意味着可能存在更一般化的对象,能把这些共有的属性和(或)方法概括在一起;
- ② 如果引入某一种更一般化的对象,那么这种对象是否反映了现实世界中更有意义的一般性;
- ③ 如果引入某一种更一般化的对象,那么这种对象是否在问题域内。

例如,当在问题域中认定了“汽车”和“飞机”两种对象,就可以把它们共有的属性和方法纳入“交通工具”这种更一般化的对象中。“交通工具”不仅在现实世界是有意义的,而且还概括了如“轮船”等其他对象。相反,“马”和“飞机”对象都有“速度”属性,如果将其概括并放置在分类结构的上层对象,则这样做不但没有实际含义,而且也失去了体现一般与特殊关系的意义。因此,类似于这种一般化的属性和方法,采用从特殊向一般的方式考虑,可对问题域中的事物抽象归纳出更为概括的公共内涵,同时不失其实际意义。

采用先从一般向特殊考虑,再从特殊向一般考虑的分类结构认定方法,是系统分析人员对于问题域的认识的深化过程。

## 2. 组装结构

组装结构又称为整体—部分结构,用于描述系统中对象之间的组成关系。它用于

描述某个对象是另外一些对象的组成部分。

如果对象 A 是对象 B 的一个组成部分，则称 B 是 A 的整体对象，A 是 B 的部分对象。并将 B 和 A 之间的关系称作整体一部分关系或“has-a”关系。

整体一部分结构指一个类的对象实例以另一个对象作为其组成部分，也可以理解为，一个类定义需要引用另一个类定义。组装结构的定义如下：

组装结构是把一组具有整体一部分关系的类组织在一起的结构。它是一个以类为节点，以整体一部分关系为边的连通有向图。

组装结构体现了 OOA 方法的聚合原则，它与分类结构同样重要，是 OOA 表示复杂事物的另一个重要手段。

组装结构的表示法是用组装结构连接符来连接结构中的体对象类和部分对象类的，其符号如图 6.19 (a) 所示。从三角形的顶角引出的连线连到整体对象类，从三角形底边引出的连线连到部分对象类。连接符两端的数字或字母用于表示该结构中对对象实例的“多重性” (Multiplicity)，含义是位于连接符一端的一个对象实例要求另一端的多少个对象实例与自己进行整体一部分组合。靠近整体对象类的数字表明它的一个对象实例需要连接符另一端几个对象实例来组成这个整体；靠近部分对象类的数字表明这个类的一个对象实例可以同时成为几个整体对象的组成部分。多重性有以下两种表示方法。

(1) 固定的数值或变量，表明要求对象的数目是确定或不确定的。如图 6.19 (b) 所示，一辆汽车只有一台发动机，所以标为 1。又如，一本书可以含有若干章节，标为“m”。

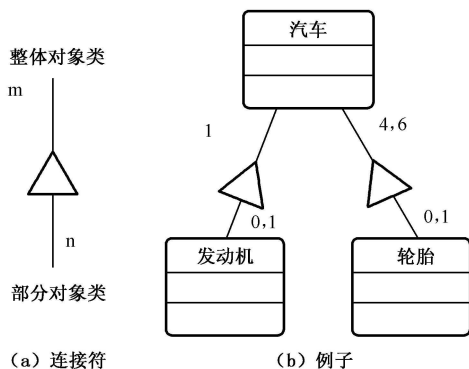


图 6.19 组装结构表示法

(2) 由固定的数值或变量组成的数对，表示要求的对象有上界和下界。如图 6.19 (b) 所示，一辆汽车有 4~6 个轮胎，用“4, 6”表示；一个轮胎可以不装到任何汽车上，也可装到一辆汽车上，所以用“0, 1”表示。又如，一个企业有 1~m 个部门，可以表示为“1, m”。

认定组装结构，总的原则是先从整体向部件考虑，再从部件向整体考虑。

对于一种对象，首先认为它是一个整体。这时，看它在问题域中含有部件的可能性，即考察：

- ① 物理上的整体事物和其组成部分，如汽车与发动机；
- ② 组织机构和其下级组织或部门，如公司与其市场部、财务部、技术部；
- ③ 团体与成员，如班级与学生；
- ④ 空间上的包容关系，如实验室与实验设备；
- ⑤ 抽象事物的整体与部分，如学科与学科分支、文章与段落等。

例如，在考察“飞机”这种对象时，找出了它的部件是引擎、机翼、座位等。如果问题域对应于引擎维护，则系统需记录的部件只有引擎，因而考虑在组装结构引入对象“引擎”，然后，确认“引擎”的每一实例都可以用属性来描述；接着检查现实世界中引擎是不是飞机的部件；最后检查是否在目标系统之内，如果目标系统是飞机引擎维护系统，则将“引擎”对象作为“飞机”对象的一个部件。可以看出，采用从整体向部件的方式考虑组装结构，是对问题域中出现的事物挖掘具体的构成细节。

在此之后，再从部件向整体考虑。对于一种对象，假定它可能是另一种对象的一个部件，可以考虑：

- ① 这种对象适合什么样的组装关系？
- ② 还需要哪些对象与这种对象一起来构成另一种对象？
- ③ 对于这样的组装而成的对象，系统是否有必要记录它的一个实例？
- ④ 这样组装而成的对象在现实世界中是否有意义？
- ⑤ 这样组装而成的对象是否限定在目标系统之内？

### 3. 实例连接

实例连接又称为链接，也称实例关联，用于表达对象之间的静态关系。这里，静态关系是指可通过对象属性来表示的一个对象对另一个对象的依赖关系。这种关系大量存在，常常与系统责任相关。例如，教师讲授课程、学生选修课程、司机驾驶车辆、顾客购买商品等都属于这种关系。在 OOA 中，用实例连接的方法表达这种关系。

实例连接是一个实例集合到另一个实例集合的映射。称为实例连接而不是对象连接，因为关联关系的两个实例集合，既可以是两种对象的实例集合，也可以是同一种对象的实例集合的子集。实例连接可分为  $1:1$ 、 $1:M$ 、 $0:1$ 、 $0:M$  四种。与 ER 图的联系相比，实例连接主要有以下特点。

(1) 实例连接是单向的。实例集合 A 到实例集合 B 的连接是  $1:M$ ，并不意味着 B 到 A 一定是  $1:1$ 。这样做的好处是既可以表示 1—多的联系，也可以表示多—多的联系。



(2) 实例连接是可选的。如果实例集合 A 中的实例必须与实例集合 B 中的一个或多个实例建立的前提下才可能存在, 则表示  $1:1$ 、 $1:M$ ; 如果 A 中的实例与 B 中的实例虽然有关联, 但 A 中实例的存在与否并不依赖这样的关联, 则表示  $0:1$  或  $0:M$  (若为  $0:1$ , 则表明有关联时相当于  $1:1$  的效果; 若为  $0:M$ , 则在有关联时相当于  $1:M$ )。

如图 6.20 所示, 图 (a) 为连接的表示法, 图 (c) 为连接表示的三种情况, 图 (b)、图 (d) 为连接的例子。

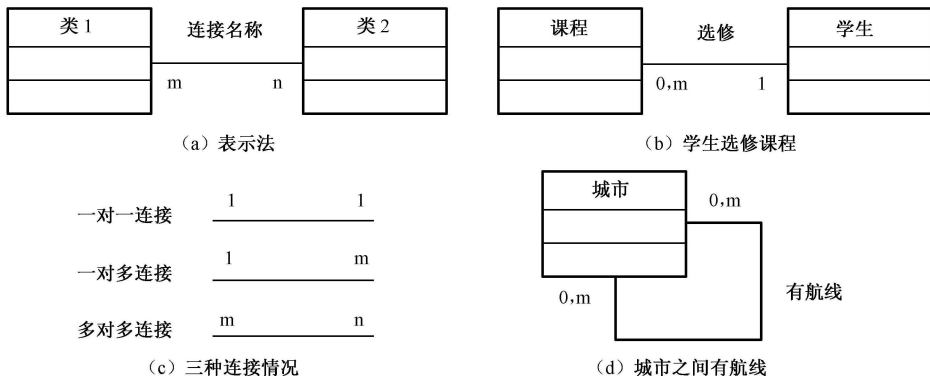


图 6.20 实例连接的例子

#### 4. 消息连接

在面向对象方法中, 消息也体现了对象行为之间的依赖关系。它是实现对象之间的动态联系, 使系统成为一个能活动的整体, 并且是各个部分能够协调工作的关键因素。如同人体的神经, 有了它, 整个机体才能活动。

对顺序系统而言, OOA 通过对消息的分析而建立对象之间的动态联系。消息体现了过程抽象的原则: 在一个对象的服务中通过消息而引用其他对象的服务。OOA 通过定义消息而把所有的对象服务贯穿在一起, 在系统实现之后, 它们将在一个控制线程中顺序地执行。

在并发系统中有多个任务并发执行。开发这种系统的难点在于: 这些任务所包含的操作在执行时没有固定的时序关系, 容易发生与时间有关的错误, 而且这些错误在调试时往往是难以再现的。因此, OOA 对并发系统的行为分析的关键的问题是把所有的对象服务组织到一些彼此并发执行, 但内部不再存在并发问题的控制线程中。

在每个控制线程内部, 完全按与顺序系统相同的方式定义对象之间的消息, 同时把解决并发问题的注意力集中于分析各个控制线程之间的消息通信上。控制线程内部的消息连接将使系统中的每个控制线程呈现出清晰的脉络。控制线程之间的消息连接



则集中地体现了系统在并发运行时各个控制线程之间的关系, 由于这种消息是不同控制线程之间的唯一联系方式, 所以并发问题能不能得到妥善解决, 焦点就集中在能不能正确地分析这些控制线程之间的消息上。

顺序系统中的消息比较简单, 并发系统中的消息则有许多不同的情况。OOA 对消息的考虑, 重点是系统责任对消息的不同要求。作为一种既适应顺序系统, 又适应并发系统的 OOA 方法, 应该识别和表示的主要问题包括:

- ① 对象之间是否存在某种消息?
- ② 这种消息是同一个控制线程内部的还是不同控制线程之间的?
- ③ 每一种消息是从发送者的哪个服务发出的? 是由接收者的哪个服务响应和处理的?
- ④ 消息是同步的还是异步的?
- ⑤ 发送者是否等待消息的处理结果?

消息连接的定义是:

消息连接是 OOA 和 OOD 模型中对对象之间行为依赖关系的表示, 即若类 A 的对象在它的服务执行时需要向类 B 的对象发送消息, 则称存在着从类 A 到类 B 的消息连接。

两个类的对象之间可能互相发消息, 例如, 在类 A 和类 B 之间, 既有从类 A 发送给类 B 的消息, 也有从类 B 发送给类 A 的消息。在同一个对象内部的不同服务之间也可能需要传送消息。

#### (1) 建立控制线程内部的消息连接

此活动的基本策略是“服务模拟”和“执行路线追踪”, 其具体做法是从类图中每个对象的主动服务开始, 做下述工作。

① 人为地模拟当前对象服务的执行过程, 考虑为了完成当前的工作, 需要请求其他对象(或本对象)提供什么服务。每当发现了一种新的请求, 就是发现了一种新的消息。

② 分析该消息的发送者与接收者在执行时是否属于同一个控制线程, 可从几个不同的角度去判断:

- 按问题域的情况和系统责任的要求, 二者应该顺序地执行还是并发地执行?
- 从发送者的执行到接收者的执行是否引起控制线程的切换?
- 接收者是否只有通过当前这种消息的触发才能执行?

③ 在当前服务的详细说明中指出由它发出的每一种消息的接收者(给出类名和服务名); 从当前服务所在的类出发, 向所有接收消息的对象类画出消息连接线(一条消息连接线可代表多种消息)。上述工作一直进行到当前的服务模拟执行完毕为止。

④ 沿着控制线程内部的每一种消息追踪到接收该消息的对象服务, 重复进行以上的工作, 这种追踪可按宽度优先或深度优先的原则, 进行穷举式的搜索, 直到已发现的全部消息都经历一遍。

当从每个主动对象服务开始的这种服务模拟和执行路线追踪都进行完毕时,对全系统中的对象类做一次检查,看是不是每个类的每个服务都曾经到达并模拟执行过。如果某个服务从未到达,则有两种可能:一种可能是这个服务是多余的,另一种可能是遗漏了向这个服务发出的消息。找出在何处发生了遗漏,加以补充。确实无用的服务应该删除。

### (2) 建立控制线程之间的消息连接

此项工作仅仅在并发系统的分析中需要。上面介绍的对象服务模拟和执行路线追踪,在找出各个控制线程内部消息的同时,可能也发现了一些控制线程之间的消息,但可能不完全。从而需要进行更全面的分析。

由于已经找出了各个控制线程内部的消息,因而可使分析人员以这些源于主动对象的控制线程作为并发执行单位,对整个系统的动态执行情况进行全局的观察,从而发现这些控制线程之间需要哪些消息。对每个控制线程,主要考虑以下几个问题。

① 它在执行时,是否需要请求其他控制线程中的对象为它提供某种服务?这种请求由哪个对象发出?由哪个对象中的服务进行处理?

② 它在执行时是否要向其他控制线程中的对象提供或索取某些数据?

③ 它在执行时是否将产生某些对其他控制线程的执行有影响的事件?

④ 各个控制线程的并发执行,是否需要传递一些同步控制信号?

⑤ 一个控制线程将在何种条件下中止执行?在它中止之后将在何种条件下由其他控制线程唤醒?用什么办法唤醒?

上述问题将从不同的角度启发分析人员去发现不同控制线程之间的消息。进一步考虑的问题是:这个消息由一个控制线程中的哪个对象服务发出?由另一个控制线程中的哪一个对象服务来处理?

根据对上述问题的思考与回答,在相应的类符号之间画出用虚线箭头表示的消息连接符。进一步分析,消息应该是同步的还是异步的,以及发送者是否等待消息的处理结果。分别在发送者和接收者的类描述模板中针对有关的服务作该消息的详细说明。

### (3) 对象分布问题及其对消息的影响

在面向对象的开发中,系统功能分布与数据分布将通过对象的分布而体现。如何把系统中的对象分布到联网的各处理机上,在 OOA 阶段还不能最终确定。因为它涉及选用什么软、硬件配置的问题,需要在设计时根据具体的实现条件来最终确定。但是分布问题又不纯粹是设计时的问题,它在很大程度上也属于系统需求。实际上,很多系统在项目可行性论证或需求报告中就已经提出了一些对系统功能及数据的分布要求。这些来自用户或经过专家论证的系统分布要求未必涉及很多具体的实现条件,无论用什么方案来实现,都应该予以满足。所以对分布问题应该从两个方面来看,一方面是系统需求提出的分布要求,另一方面是设计决策问题。OOA 应该只考虑前一方面的问

题，后一方面的问题则应推迟到 OOD 中考虑。

分布问题对 OOA 的影响如下所述。

① 在每台处理机上分布的一组对象中，至少应有一个对象是主动对象；所有的被动对象都是在位于本机的主动对象的驱动下运行的。如果这一条不满足，则应考虑增加主动对象，或把其中的某些被动对象改为主动对象。

② 同一台处理机上的对象之间的消息通信可能是一个控制线程内部的，也可能是不同控制线程之间的，分布在不同处理机上的对象之间的消息通信只能是不同控制线程之间的。它们在实现时所依赖的消息通信机制也有所不同。例如，前者可用函数调用或 IPC 来实现，后者则需使用 RPC 实现（实现技术不必由分析人员考虑，但了解这种背景知识对分析人员更确切地认识消息是有益的）。

根据系统需求中已经明确的分布要求，分析人员可把 OOA 模型中的对象类进行初步的分组。设想每一组对象是分布在一台处理机上的，从而确定哪些消息属于本机的通信，哪些消息属于不同处理机之间的通信。必须注意的一点是，在未考虑对象分布问题之前，我们可能把某两个对象之间的消息看作一个控制线程内部的消息，对接收者的服务请求如同一个顺序执行的函数调用。现在，如果这两个对象分布到不同的处理机上，则接收者就不能以这种方式提供服务，它与发送者必须属于不同的控制线程。那么，接收者是在哪个主动对象的驱动下对外响应请求并提供服务呢？如果缺少这样的主动对象，则考虑：是把某个被动对象改为主动对象还是增加一个主动对象？

然后分三种情况定义对象之间的消息：

- ① 本地机上同一个控制线程内部的消息；
- ② 本地机上不同控制线程之间的消息；
- ③ 异地机上不同控制线程之间的消息。

对象分布问题一般不能在 OOA 阶段完全确定，而要在 OOD 中按实现时的软、硬件配置进一步考虑。如果在 OOA 中不能确定某些对象是否分布到不同的处理机上，则可暂时把它们看作是在同一台处理机上的，留待 OOD 中作进一步的处理。

消息的详细说明包括对发送者和接收者两方面的说明。在发送者的类描述模板中对每个发送消息的服务作如下说明：

- ① 指出这个服务在执行时可能发出的每一种消息，给出接收者的类名和处理该消息的服务名；
- ② 说明接收者与本服务是顺序执行还是并发执行；
- ③ 必要时说明该消息是同步的还是异步的，以及发送者是否等待该消息的处理结果；
- ④ 如果服务流程图是比较详细的，则应画出在什么位置上发送什么消息。

在接收者的类描述模板中对每个服务作如下说明：

- ① 说明由这个服务接收和处理的每一种消息，规定消息的格式及内容（称作消息协议），内容包括消息名称、输入/输出参数及参数类型；
- ② 说明本服务是与发送者顺序执行的还是并发执行的。

仍以前述的商场销售管理系统为例。

### （1）分类结构

一般类“商品”和它的两个特殊类“特价商品”及“计量商品”构成分类结构。在这个结构中，一部分属性和服务是多态的。

### （2）组装结构

“商品一览表”和“商品”构成一个组装结构，通过前者的“商品目录”属性体现这种关系。

“账册”和“销售事件”构成另一个组装结构，通过前者的“销售事件表”属性体现这种关系。

### （3）消息连接

对这个系统，我们暂且不考虑各个消息连接是控制线程内部的还是控制线程之间的，首先把所有的消息连接都找出来。从收款员、供货员和上级系统这三类活动者的相关对象开始执行路线追踪，以发现系统中的各种消息连接。

“收款机”对象在执行“售货”服务时向“商品一览表”对象发消息，请求其“检索”服务以找出相应的“商品”对象；接着，它要向“商品”对象发消息以获知该种商品的属性信息并执行其“售出”服务；此时若“商品”对象发现该种商品数量低于规定的下限，则向“供货员”对象发消息进行“缺货登记”。“收款机”对象向“销售事件”对象发消息，请求“销售计价”和“入账”等服务；在执行“入账”服务时，“销售事件”对象向“账册”对象发消息以执行其“记账”服务。“收款机”对象在执行自己的“登录”服务时，向“账册”对象发消息，请求“接班”服务。“收款机”对象在执行“结账”服务时，也要向“账册”对象发消息，请求“交班”和“报账”两个服务；此时，“账册”对象要向“上级系统接口”对象发消息，通过它的“报账”服务向上级系统报账。

“上级系统接口”对象在上级系统要求查账时向“账册”对象发消息，请求它的“报账”服务；在上级系统要求进行价格更新或商品种类增删时，分别向“商品”和“商品一览表”对象发消息，请求相应的服务。

“供货员”对象在执行“供货”服务时向“商品”对象发消息，执行其“补充”服务。

假如按用户需求确定的系统分布方案是：把仅与一台收款机处理的业务有关的信息与功能集中在一起作为一个客户机，把所有的客户机共享的信息集中在一起作为服务器。那么，“收款机”、“销售事件”和“账册”三类对象将分布在一起（每台客户机

上都有这样一组），其余六类对象将分布在一起（在服务器端）。

按这样的分布要求可以确定，从“收款机”发送到“商品一览表”和“商品”的消息，以及“账册”和“上级系统接口”之间互相发送的消息，都是不同控制线程之间的消息，其余的消息都是各个控制线程内部的消息。

综上所述得到该系统 OOA 模型的关系层并构成整个类图，如图 6.21 所示。

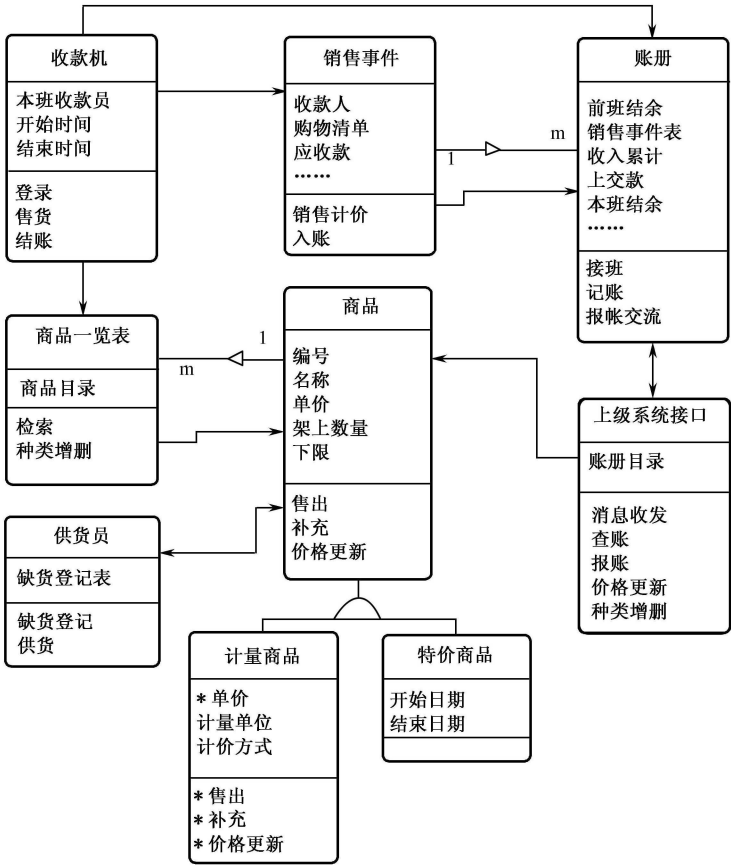


图 6.21 商场销售管理系统结构关系层

6.3.7 定义主题

1. 主题的概念

开发一个实际的系统，通常不会像书本上用来举例的系统那样简单，其 OOA 模型中的类可能要达到较多的数量。一个类图中的类如果有几十个，则人们对它观察和理

解时就会感到困难。面对这么多的类，以及它们之间错综复杂的关系，阅读者会感到茫茫一片，不得要领。根据心理学家 G. Miller 于 1956 年得出的一项研究结果：人同时可以考虑和理解的问题数目受到记忆能力和处理能力的制约，一般不超过  $7 \pm 2$  个。因此人类在认识复杂事物时，学会了采用粒度控制的原则，即当从系统全局考虑问题时，重点着眼于系统中那些高层次、大粒度的概念进行宏观的思考，暂时不细察其内部细节，以免“明察秋毫之末，而不见舆薪”。在考虑各部分的细节时，则集中于一个局部，围绕一个主题进行微观的思考，暂时撇开当前考察范围以外的其余部分。

为了能控制系统的复杂性，OOA 方法也支持这种思考方式。其基础是把一个大的、复杂的系统按其固有的特征与规律，组织成从宏观到微观的多个层次。最高的层次是为数不多的几个大主题；每个主题由一些对象类或较小的主题构成；每个对象内部由其属性和服务构成。为此，需要引进一种比类和对象的抽象层次更高、粒度更大的概念，用以建立系统的高层抽象视图。这种概念称为主题，其定义如下：

主题是把一组具有较强联系的类组织在一起而得到的类的集合。

按照上述定义，主题的概念具有以下特点。

(1) 它是由一组类构成的集合，但它本身并不是一个类（开发者不必像定义一个对象类那样指出它有什么属性和什么服务）。

(2) 一个主题内部的对象类应具有某种意义上的内在联系。例如，这组类一同描述了系统中某个相对独立的组成部分（如一个子系统），描述系统中某一方面的事物（如人员或设备），或者解决系统的某一方面的问题（如输入、输出）等。因此，每个主题内部应该是高内聚的，各个主题之间应该是低耦合的。

(3) 主题的划分有一定的灵活性或随意性，可以“仁者见仁，智者见智”。强调的重点不同可以得到不同的主题划分，每一种划分可能都是合理的。

OOA 对主题的使用有两种方式。一种是自底向上的，即先建立类图，然后把类图中每一组联系较强的类组织为一个主题。如果主题数量仍然太多，则进一步把联系较强的小主题组织成一个大主题，直到系统中最上层的主题数量控制在七个左右。这种方式适应于小型系统或中型系统。另一种方式是自顶向下的，即先对系统做初步的分析，确定几个大的主题，每个主题相当于一个子系统，按这些子系统进行分工，各个分析小组对自己分工的子系统进行正规的面向对象分析，建立各个子系统（主题）中的类图。最后各个小组的分析结果将合并为一个大的 OOA 模型。这种方式适应于大型系统。

无论采用哪种方式，最终的结果将是：一个完整的类图作为系统的基本模型，一个主题图作为系统的补充模型。

有了主题图，当人们要观察一个 OOA 模型时，可以首先看到最上层的主题（其数量不超过  $7 \pm 2$ ），由此可以简明扼要地了解到系统的宏观组成情况，知道它包括几个大



的组成部分。如果要了解某个主题的内部情况，则可以把它半展开，此时可看到这个主题中包含了哪些类和哪些下一级的主题，这些类和下级主题的总数也不超过  $7 \pm 2$  个。如果要进一步了解这个主题中每个类的内部特征及彼此之间的关系，就把这个主题全展开，此时将看到这个主题范围内详细的类图内容（包括类符号、其属性与服务，以及类之间的结构与连接）。一个主题内部嵌套的下级主题也可以这样半展开或全展开。当系统中全部的主题及嵌套的下级主题都被完全展开时，就是一幅完整的、详细的类图了，这个类图上带有一些框，每个框就是一个主题。

人们也可以利用主题自底向上地观察 OOA 模型。在完全展开的类图上利用主题框找到自己所关心的一个局部，了解其中的类及其相互关系（如果不希望旁边那些不感兴趣的内容干扰视线或占据屏幕，则可以把它们收缩到各自的主题中）。对下层的内容已经掌握时，把它们收缩为一些较高层的主题。最后，在主题图的最上层了解系统的总体情况。

在 OOA 工具的帮助下，主题的展开与压缩可以很方便地进行，如同摄像机镜头的聚焦与缩放一样，可以使人们既可以看到一幅粗略的全景，又可以看到一个详细的局部。

综上所述，主题对于 OOA 的作用是：为分析人员提供了一种比类和对象层次更高、粒度更大的抽象手段，使他们可在 OOA 基本模型（类图）之上建立多层次的系统视图。因此，它可使分析员在不同的抽象层次上来认识和描述问题域。主题对于 OOA 模型的阅读者（包括设计、编程、测试、维护人员、用户、管理者等，也包括分析人员自己）的意义是：提供了一种控制系统复杂性、引导读者有条理地观察 OOA 模型的机制，使他们可以在不同层次上对 OOA 模型作概要的或详细的理解，并使他们在一段时间内把需要理解和记忆的信息控制在智力可承受的合理范围内。

## 2. 主题的表达方法

主题有三种表示方式——压缩方式、半展开方式和全展开方式。

压缩方式的主题表示符号是一个矩形框，内部填写主题的编号和主题名，其中编号应该体现主题的层次。最上层的主题编号为 1, 2, 3, …；1 号主题的下层主题编号为 1.1, 1.2, 1.3, …；余此类推。主题名应该使读者能顾名思义地了解这个主题所包含的内容。

半展开方式的表示法是在矩形框的上栏内填写主题的编号和主题名（其要求与压缩方式相同），下栏内列出这个主题中所包含的类和下层主题（只列出其名称）。这里列出的类是那些不属于下层主题的类。下层主题中包含的类只有当下层主题展开时才能列出。类名尽可能以缩进的格式排列，以体现其间的继承关系或组成关系。

压缩方式和半展开方式的主题表示法在构造主题图时使用。一个主题图可以全部由压缩式的主题表示符号构成，或全部由半展开式的主题表示符号构成，也可以两种表示方式混合使用。

全展开方式的主题表示法是在类图上用一个多边形框出这个主题所包含的类符号，并在凸角位置上标出主题的编号；其内部是类图上原有的全部内容（类符号及它们之间的结构与连接表示符号）。用这种方式表示的主题不是独立存在的，只有画到类图上才有实用意义。主题的压缩、半展开和全展开表示如图 6.22 所示。

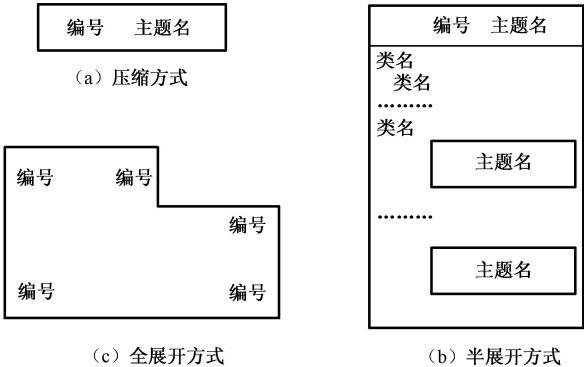


图 6.22 主题的压缩、半展开和全展开表示方式

如上所述，划分主题的活动可以是自顶向下的，也可以是自底向上的。如果系统规模较大，需要由多个小组分头进行分析工作，则较合理的策略是先由高层人员对系统作宏观的了解，划分出一些大的主题，并把每个主题作为一个子系统分配到各个分析小组。如果系统规模可由一个分析小组集中地完成，则上述步骤可以省略。以下的工作对于两种情况都是共同的：建立系统（或子系统）的类图，然后在类图上划分主题，最终形成系统的 OOA 补充模型——主题图。

3. 主题的划分

主题的划分即把类图中的类划分到一些最低层的主题中。类图中的每个分类结构和每个组装结构都是一组联系较为紧密的类，所以首先考虑把每个结构作为一个主题。

(1) 对于分类结构，可以用这个结构中最上层一般类的类名作为主题名（但不是绝对的），主题中包括这个结构中的所有特殊类。在多继承的一般特殊结构中，可能有多个最上层的一般类。对于有少量交叉的两个结构，可以划分为两个主题；对于交叉部分太多而很难看成两个结构的，可以划分到一个主题，并为它取一个合适的主题名，如图 6.23 所示。

(2) 对于组装结构，可以用这个结构中最上层整体类的类名作为主题名（但不是绝对的），主题中包括这个结构中的所有部分类。最上层的整体类如果有多个，则也根据交叉部分的多少决定是否划入一个主题。



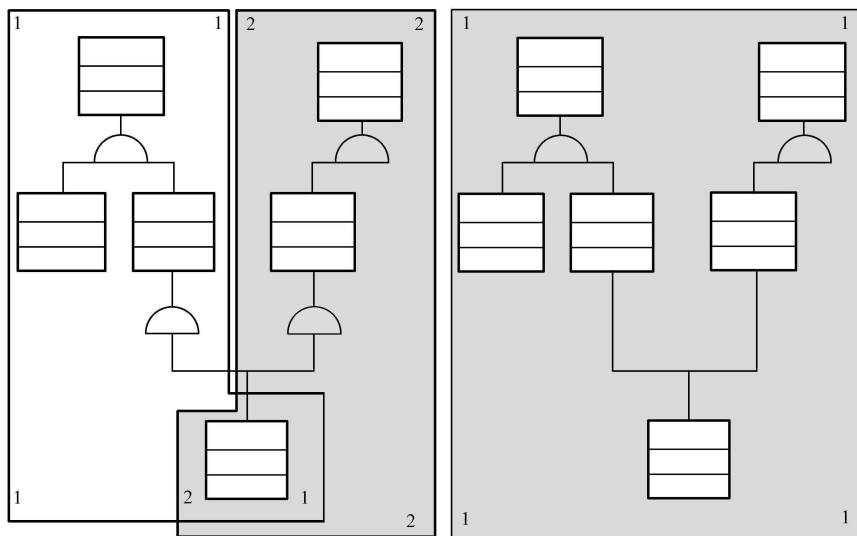


图 6.23 一般特殊结构主题划分

通过实例连接互相联系的类可考虑划分到一个主题中。如果连接线一端的类已经进入一个主题，则可把另一端的类也带进这个主题中。

(3) 最后剩下的是一些既不属于任何结构，也没有实例连接的类。暂时把每个这样的类作为一个主题。

通过以上的划分，如果主题的数量已在  $7 \pm 2$  之内，则划分主题的工作可到此为止。如果这些低层主题的数量较多，则进行主题的合并。

#### 4. 主题的合并

主题的合并工作是把一些较小的主题合并为一个较大的主题。下层主题被合并之后，可以取消，也可以保持。最终应使最上层的主题数量不超过  $7 \pm 2$ 。为判断哪些主题应该合并，可考虑以下几个方面的因素。

(1) 根据问题域的情况，如果某几个主题所包含的对象类在概念上比较接近，或者有较强的相关性，则可考虑合并为一个主题。例如，“人员”主题和“组织”主题，它们分别包含系统中有关各种人员和各种组织的对象类。在现实世界中，人员与组织是很容易引起联想的两组概念，并且有很强的相关性，所以可把它们合并为一个主题，命名为“人员”或“人事”。

考虑问题域的另一种方式是自上而下地看问题域现行运作体系的划分。例如，一

所大学有教务、科研、行政、后勤等几个运作体系，据此可把与它们有关的下层主题合并到这样一些大的主题中。

(2) 考虑系统责任，如果某几个主题所涉及的系统责任有较大的相关性，或者说，它们的功能同属某项大的功能，则可考虑合并为一个主题。例如，在商场管理系统中，进货管理、销售管理和库存管理是系统责任的几个大的方面，可把与它们有关的小主题合并到这三个大主题中。

(3) 在类图上观察各个（全展开的）主题之间的关系强弱，把强耦合的（即各种关系较多的）主题合并为一个主题。有以下几种情况。

- 主题之间有交叉，即有一个（或几个）类同时属于多个主题。
- 主题之间有结构连线。如果一个类是一个以上的结构中的成分，而在划分低层主题时只把它划入其中一个结构所在的主题，则这个类与其他结构的连线将跨越主题的边界。如果主题之间有较多的结构连线，则说明它们之间有较强的联系应考虑把它们合并。
- 主题之间有较多的实例连接线或消息连接线。这种情况同样说明主题之间有较强的联系，可考虑合并。

(4) 在分布式应用系统中，可参考系统功能的分布情况进行主题的划分与合并。分布在同一种节点上的几个较小的主题可合并到同一个大主题中。

通过合并，主题的数量减小了。如果最上层主题的数量仍然较多，则进一步按以上策略对高层主题进行合并，直到其数量减少到  $7 \pm 2$  以内。

主题合并之后，如果低层主题仍然保留着，则它嵌套于高层主题之中，形成了层次。主题层次的存在可以使 OOA 模型的阅读者灵活地控制自己的视野。但如果主题嵌套的层次太多，也将增加模型的复杂性，进而影响对它的理解。所以，主题嵌套的层次不宜太深。控制主题嵌套深度的策略是：对嵌套的主题进行审查，尝试把某一层的某些主题取消，即取消某些主题的边界线，使其内部成分暴露到它的上一层。这种尝试应始终坚持下述原则：系统中最高层主题的数量和每个主题内部的成分（类或者低层主题）数量都不超过  $7 \pm 2$ 。如果取消某个主题的尝试将违犯这一原则，则应放弃这一尝试。

通过对主题层次的控制，许多中小型系统可以只设一层主题，最多不超过两层；许多大型系统可以只设两层主题，最多不超过三层。

商场销售管理系统的规模不大，本节中所讨论的某些有关主题的复杂情况在这个系统中都没有出现，而且在工程实践中对于这样不超过十个的系统即使不划分主题也可以一目了然。这里只想通过这个例子将本节的内容与实际系统联系起来。

先识别低层主题，然后进行合并，最终得到以下三个主题。

“收款机”、“销售事件”和“账册”三个类组成一个主题，取名为“销售记录”，编号为 1。“商品一览表”、“商品”、“特价商品”和“计量商品”四个类组成一个主题，取名为“商品信息”，编号为 2。“上级系统接口”和“供货员”两个类组成一个主题，命名为“外部接口”，编号为 3。

按照这样的划分，在系统的类图上画出全展开方式的主题表示，如图 6.24 所示，压缩方式和半展开方式的主题图如图 6.25 所示。

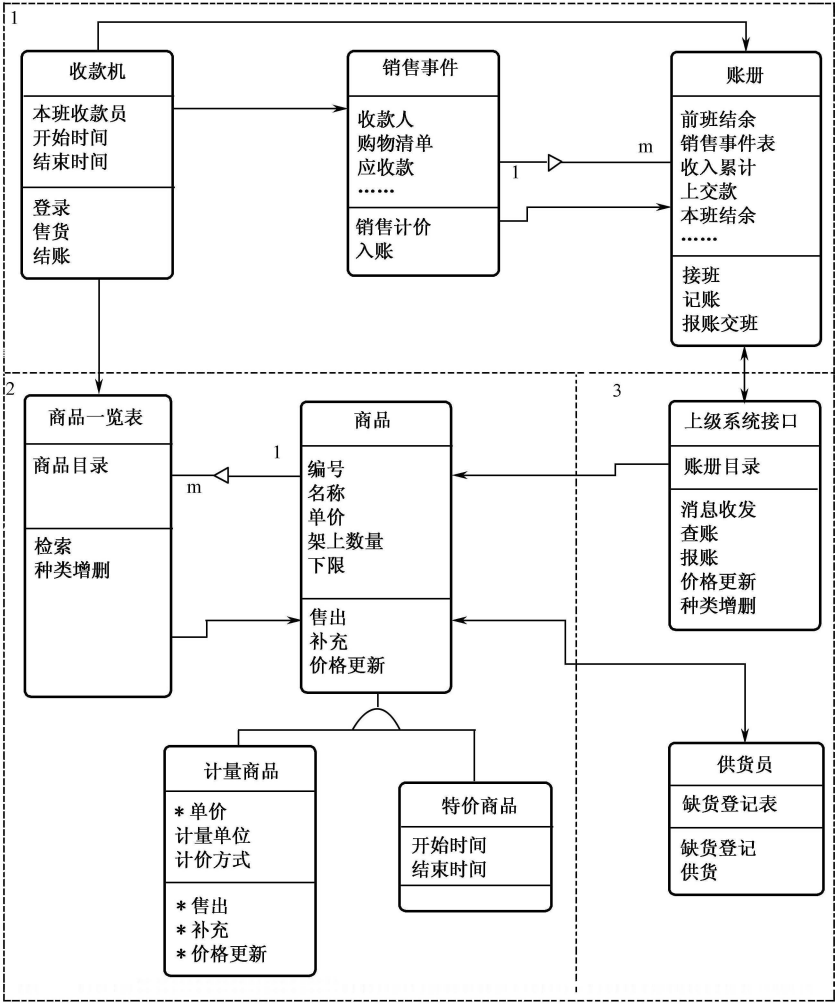


图 6.24 商场销售管理系统的主题图

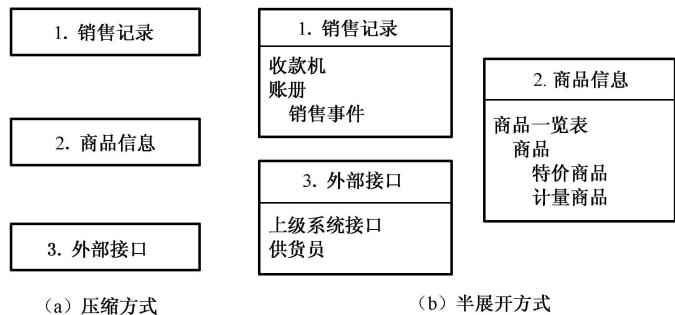


图 6.25 商场销售管理系统的全展开主题图

6.4 面向对象的设计

在 OOA 阶段中，分析主要针对问题域和系统责任，不考虑与实现有关的因素。OOA 模型由五个层次组成，即主题层、类及对象层、结构层、属性层和方法层，对应着分析工作的五个活动。OOA 得到的分析结果模型化了“问题空间”。

从 OOA 到 OOD（Object Oriented Design）是一个渐进的模型扩充过程。在 OOD 的过程中，需要继续运用 OOA 中的原则与方法（OOA 的五个活动），采用一致的表示方法，从问题域、人机交互、任务管理和数据管理四个部分出发（如图 6.26 所示），针对实现的要求进行必要的增补和调整，例如，需要对类、结构、属性及方法进行分解和重组。这种分解是根据一定的过程标准来做的（标准包括可重用的设计与编码类），把问题域专用类组合在一起，通过增加一般类来创立约定，提供一个继承性的支撑层次来改善界面，提供存储管理，以及增加低层细节等。

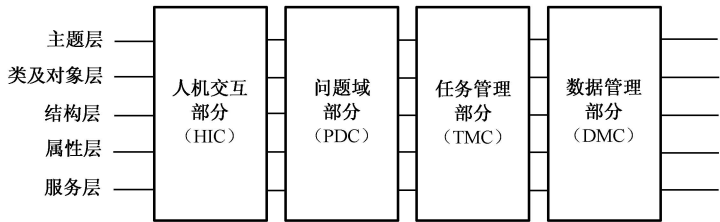


图 6.26 四个部分

问题域部分在 OOA 阶段已经进行了详细的分析，对于在 OOA 阶段未曾考虑的其他三个部分，全部在 OOD 阶段建立。

(1) 人机交互部分 (HIC)，根据用户选用的图形用户界面系统 (GUI) 和特定用户对人机界面的要求而设计的系统使用界面。它在很大程度上依赖于所用的图形用户界面环境 (如 MS-Windows、Linux 等)，由新定义的人机界面类和对象组成。

(2) 任务管理部分 (TMC)，用于定义系统中需要并发执行的各个任务，包括任务的定义、通信和协调，以及硬件分配、外部系统及设备约定。可能包括的类有“任务”类和“任务协调”类。

(3) 数据管理部分 (DMC)，按选定的数据管理系统而设计的负责对象的存储及检索的系统组成部分。它与物理的数据管理方法无关，可以是普通文件、带标记语言的文件、关系型数据库、面向对象数据库等。可能包括的类有“存储服务”类，协调每个需永久保存的对象的存储。

OOA 和 OOD 使用统一的表示方法，在二者之间并没有一个十分清晰的分界线。OOA 阶段所作的工作在 OOD 中可以直接应用，OOD 的工作又可能对 OOA 的工作产生一定的影响。在设计开始之前，分析不必圆满地完成，这样是符合人类认识世界和改造世界的基本过程的。因为一旦认识到事物是复杂多变的，而又不能无所不知时，就会看到“圆满”完成任何事情的可能性都很小。

#### 6.4.1 问题域部分的设计

问题域部分由与问题域有关的对象构成，并且在特定的实现条件 (如编程语言) 下提供用户所需功能的组成部分。它是在 OOA 分析模型的基础上按实现的要求进行必要的修改、调整和补充得到的。可见，OOD 是在 OOA 基础上的扩充，其表示方法和含义是一致的，没有本质的差别。原因如下。

首先，从 OOA 到 OOD 不是一个突变的过程，而是在对 OOA 的结果作深入分析的基础上，结合具体的实现环境和要求，对 OOA 的结果的改动和增补，这是一个循序渐进的过程。OOA 模型忽略了系统的实现条件，只考虑问题域和系统责任，抽象层次较高；OOD 模型中的问题域部分则在选定的实现条件下对 OOA 模型进行具体化，抽象层次较低。

其次，OOA 改动和增补的结果是在设计 (Design) 方面、编程 (Programming) 方面与问题域之间的转换尽可能相近而形成的一种修改准则，即为对于一个特定问题的设计要考虑其需要的实际变化。这些扩充设计的内容可能需要分解或重组一些类及对象、结构、属性、服务，而且仅当基于特定的客观标准 (如语言限制) 时才合乎情理。

面向对象的范型 (Paradigm) 是按照问题本身组织系统框架。它从分析到设计再到编程这个过程采用的思想方法和表示方法是极为相近的。在实际中，用户的需求总是处于不断的变化中，这就要求系统也必须能够适应不断变化的要求。于是，如何保

持系统结构的相对稳定就显得格外重要了。从面向对象的角度而言,分析、设计、实现过程中的思想、方法和技术都是相近的,不同点在于重点考虑问题的方面各有所侧重,因此,分析、设计、编程结构具有长期的稳定性。虽然细节部分会随着用户的需求而发生变化(如增加类、属性或服务),但基于问题域的总体结构框架可以保持稳定。这种稳定性使得问题域相似的系统之间的分析、设计及编程结果可以重用,也为一个系统超出其生命期应具备的可扩充性(即增加和减少其他功能)提供了保障。

问题域部分的设计要针对特定的实现环境,对 OOA 的结果加以增补,要考虑以下因素:

- (1) 针对编程语言支持能力进行调整;
- (2) 增加一般类,提供共同协议;
- (3) 为实现复用采取的设计策略;
- (4) 提高性能;
- (5) 提供数据管理部分;
- (6) 完善对象细节。

#### 1. 针对编程语言支持能力进行调整

不同程序设计语言对面向对象概念的支持程度有所不同。例如,目前许多面向对象的语言(Object Oriented Programming Language, OOPL)不支持多重继承和多态,而非面向对象的语言一般支持结构化程序设计,因此设计时必须按照 OOPL 的特点进行调整。

对象、类、属性、服务、消息、关联、聚合、封装、继承等 OO 概念是大多数 OOPL 都能支持的。但是有的 OOPL 仅支持单继承而不支持多继承。多态性也只有部分 OOPL 能够支持。永久对象和主动对象的概念,目前多数 OOPL 都不支持。不过前者可通过数据接口部分的设计来解决,后者可利用语言的主函数和操作系统的并发功能来实现,无须对问题域部分做明显的调整。

一般—特殊结构和整体—部分结构在有些情况下可以互相变通。其原理是:尽管继承和聚合反映了现实世界中两种不同的关系,但从最终效果来看也存在着共性,即都是使一个类的对象能够拥有另一个(或另一些)类的属性和服务。OOPL 通过不同机制实现继承和聚合,对继承支持有强有弱,对聚合的支持却是都能提供的。甚至用非 OO 编程语言也能清晰地实现聚合。根据以上说明,可以在 OOPL 不支持多继承的情况下,运用聚合化解多继承,将多继承的一般—特殊结构转化为单继承与聚合关系的混合结构,或者转化为只含聚合、不含继承的整体—部分结构。例如,如图 6.27 所示。

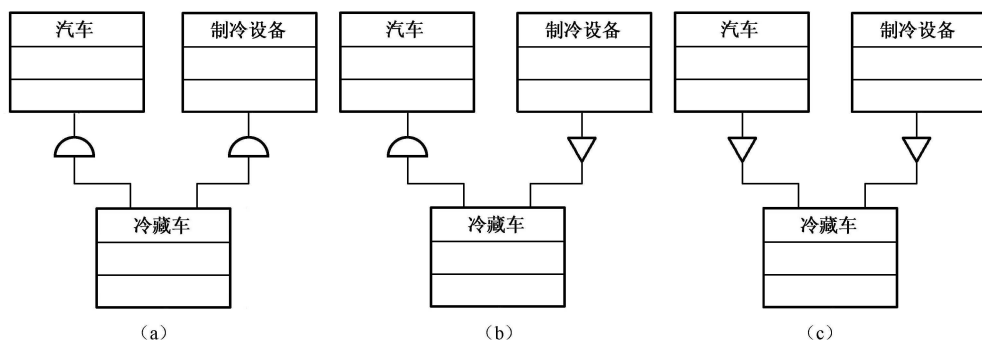


图 6.27 多继承的转换方法

图 6.27 (a) 所示为多继承结构，冷藏车分别从汽车和制冷设备两个对象中继承属性和方法，转化为其他结构如图 6.27 (b)、(c) 所示。其中图 6.27 (b) 中，冷藏车从汽车对象继承（一般—特殊结构），制冷设备作为冷藏车的组成部分（整体—部分结构）；图 6.27 (c) 则将汽车和制冷设备均作为冷藏车的组成部分（整体—部分结构）。在图 6.27 的例子中转换是合适的，但在图 6.28 中则不合适。

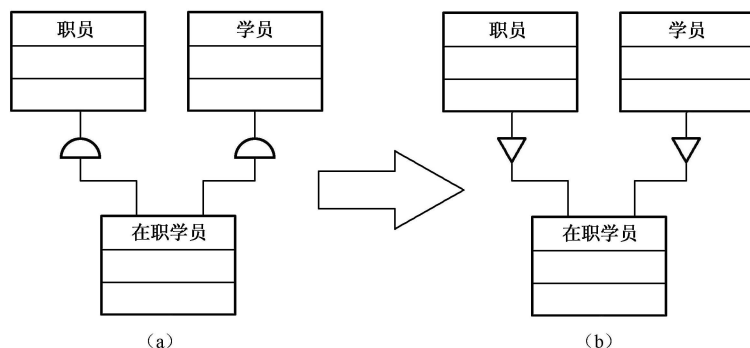


图 6.28 不合适的转换

在多重继承的情况下没有问题（图 6.28 (a) 所示），转换成为图 6.28 (b) 后，含义变成了在职学员由职员和学员两个部分组成，显然是不正确的。解决的方法是，可以通过定义新的类来完成，如图 6.29 所示。其中图 6.29 (b) 增加了两个身份对象分别描述职员身份和学员身份，通过聚合使人员要么是职员，要么是学员或者是在职学员。图 6.29 (c) 则另外增加了“身份”对象作为职员身份和学员身份的一般类，“人员”对象“聚合”身份对象。这样也可以达到清晰描述问题的目的，同时将多重继承转换为单继承。

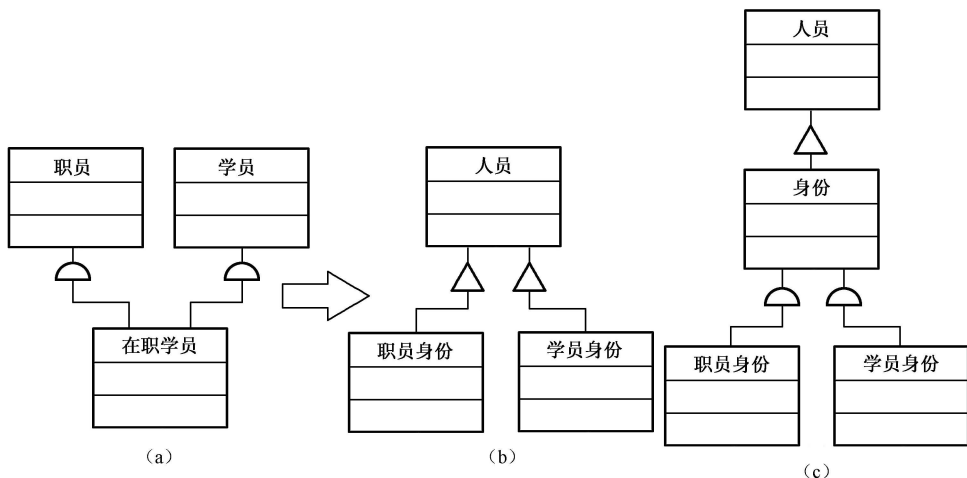


图 6.29 合适的转换

目前，常用的程序设计语言比较多（如 C/C++、Java、Object Pascal、C#、Visual Basic、PowerScript 等），它们支持面向对象的程度各有差异，其中 C++ 语言支持面向对象的能力最强，其他次之。在设计时，当选定了编程语言之后，就必须根据该语言提供的面向对象功能的强弱对 OOA 的分析结果进行适应性改造，改造的方法可以多种多样，但是无论如何调整，得到的 OOD 结果针对问题域和系统责任方面与 OOA 必须是一致的，否则将偏离目标而导致最终系统功能不能满足用户的需求。

## 2. 增加一般类，提供共同协议

前面曾提到，对象的创建、删除、复制、转存等行为实际上是系统行为而不是个别对象自身的行为。各种 OOPL 往往可在某种程度上统一地解决这些问题，所以在分析阶段不要求在各个对象中定义与此有关的属性和服务。在设计阶段编程语言已经确定，这些问题到底如何解决必须给出明确的答案。除了上述全局性问题之外，还有一些局部的（但并非个别的）问题。例如，模型中凡是需要永久保存对象信息的类，都需要某些共同的属性和服务；凡是需要向用户界面发送信息的类，都需要一些共同的属性和服务。此时应根据编程语言提供的支持，定义系统中全部对象类或某一部分对象类共同需要的属性和服务（即所谓的共同协议）。例如，某种语言可能提供了一个名为“Object”的类，其中定义了所有的对象都需要的属性和服务，供系统中所有的对象类继承；那么，OOD 模型中就可以明确以这个类作为最高层的一般类，使模型中所有的类都可直接地继承它。如果语言没有这样一个预定义的类，则设计者应该在模型中



自己定义这样的类。

对局部性的问题也可如法炮制。例如，在所有需要永久保存对象信息的类之上增加一个一般类，提供它们共同需要的属性和服务（也可根据语言是否提供相应的支持，决定是直接利用语言的功能还是自己定义这个类）。

在 OOA 中定义一般类的主要目的是：集中地描述问题域中事物的共同特征，将多个类都具有的特征提升到一般类中进行表示。在 OOD 中定义上述一般类的主要目的是：描述特定条件下某些（全部或部分）类的共同实现策略，用一个一般类集中地给出多个类的实现都要使用的属性和服务。

### 3. 为实现复用采取的设计策略

软件复用可分为程序代码级、设计级和分析级等不同级别。相应地，可复用构件也分为相应的级别。在开发过程中的各个阶段充分利用不同级别的复用构件，可以显著提高系统的开发效率和质量。在 OOD 阶段应尽可能地复用设计构件，并为代码级复用给出明确的设计表示。

目前，许多 OOP 都带有一个比较完整的类库（如 Visual C++ 的 MFC 类库）。类库中的每个类都是提供可复用的。在设计时，应充分利用语言提供的这些类库（可能是第三方提供的），从而提高设计效率和质量。

（1）直接复用：如果类库中定义的类恰好与系统的要求完全相符，则可以直接在 OOD 模型中使用这个复用的类。例如，许多类库提供动态对象数组类（如 VC++ 中 COArray 类），如果系统中某个对象中需要定义对象数组（对象为系统中的对象），则可以直接使用动态数组对象来完成，而不需要定义一个新类，只需要说明类是复用了类库中定义的类即可。

（2）通过继承复用：如果可复用的类中的属性和服务都是系统所需要的，但是并不完整，则可以通过继承来复用。例如，类库中有许多显示控件（如列表等），如果设计中有“人员统计”对象，则可以考虑从显示控件（如列表控件）继承，这样新对象中具有处理业务的功能，同时也能够提供用户的使用操作。

### 4. 提高性能

尽管计算机的计算速度大大提高了，但是对许多系统（尤其是实时系统）而言，性能也是至关重要的。系统的性能是用户要求的内容之一，一般为一些时间要求较高、提出完成一项功能的最大可容忍时间。由于性能与很多因素相关，如硬件设备、网络、数据管理系统、图形用户界面和编程语言等，因此将它放到 OOD 阶段考虑。

影响性能的因素可分为以下三个方面。

(1) 网络传输时间：不同的网络环境，对系统的性能的影响是巨大的。它包括系统分布方案、网络拓扑结构、数据传输速率和网络拥挤状况等因素。

(2) 数据存取时间：即将数据在外部存储设备上存取的时间。在不同类型的存储设备上，存取时间不同，如单个磁盘与磁盘阵列相比后者快得多；存取的数据量不同，存取时间也不同；采用的数据管理方式不同（如文件与数据库系统相比），存取时间也不同；数据存取的频繁程度不同，存取时间也不同；等等。

(3) 数据处理时间：即系统中计算的复杂性程度。尽管计算机的计算性能提高得很快，但在某些情况下，算法的实现效率也有一定的影响。

由于影响系统性能的因素很多也比较复杂，所以只有在具体问题中具体考虑，找出影响性能问题的主要因素后，才能逐步解决。下面有几种可以改进性能的措施：

(1) 在对象之间具有高度繁忙的消息流通的情况下，这种高度耦合可能需要把两个或更多的类进行合并；

(2) 在类及对象中扩充一些保存临时结果的属性；

(3) 避开正常的数据库抽象原则而允许服务从其他对象中强行获得属性值。

## 5. 提供数据管理部分

为了提供数据管理部分，每个被保存的对象需要知道自己是怎样被存储的。

(1) 第一种方法是“每个对象自己保存自己”。具体做法如下：通知对象保存自己；每个对象知道如何保存自己；增加完成此事的属性和服务。

(2) 第二种方法是，每个对象把自己传送给数据管理部分，让数据管理部分来存储对象自己。具体做法如下：通知一个对象保存自己；每个对象知道为了保存自己的状态应该传送什么消息到数据管理部分，增加属性和服务以完成此事。在这种方法中，可以把任何一个提供同一组服务的存储系统插入数据管理部分，而无须对问题域部分做额外的修改。

无论第一种方法还是第二种方法，都需要如下属性和服务。

① 对于单继承环境，直接修改问题域部分：

- 增加一个属性来标识对象属于某个特定类，如增加一个 `ClassName` 属性；
- 增加一个服务来定义对象如何存储自己的值；
- 隐式地保持它们，不是包含在图中，而是在每个类及对象的说明部分定义它们。

② 对于多继承环境：

- 把增加的放在一个新类中，然后修改每个带有可存储对象的类，使之继承这个附加的一般类；

- 增加一个属性来标识对象属于某个特定类，如增加一个 `ClassName` 属性；
- 增加一个服务来定义对象如何存储自己的值；
- 隐式地保持继承，不是包含在图中，而是在每个类及对象的说明中定义。

(3) 第三种方法是，每个需要长期保存的对象由一个面向对象数据库管理系统 (OO-DBMS) 来管理：

- 每个对象都由 OO-DBMS 来管理，而 OO-DBMS 负责为可考虑中的系统存储和检索对象；
- OO-DBMS 为维持贯穿设计和编程的问题域结构框架提供了最好的潜在可能性，但是采用其他的方法也是明智的。

## 6. 完善对象细节

OOD 设计得到的结果将直接转换为实际编程语言的对象、类定义。在 OOA 阶段已经从问题域和系统责任出发建立了许多对象的细节，这些细节是从分析角度考虑的，对设计具有很大的帮助。在 OOD 阶段，设计工作需要进一步细化，它将给出一个完全可实现的 OOD 模型，凡是需要让程序员知道的对象，包括对象的每一个属性、服务和其他必要的细节，都应该在 OOD 模型中说明或定义清楚。

(1) 弥补 OOA 模型的不足：从 OOD 的设计角度出发，针对问题域中的对象，检查它是否具备了表达问题域和系统责任所需的所有属性和服务；针对每个类的每个属性和服务，检查它的定义是否完整。

(2) 解决 OOA 阶段推迟考虑的问题：在 OOA 中未能考虑的问题，应在 OOD 中设计完善。一方面，按照严格的封装原则，对象内部不能为外部所见，而必须通过服务来实现访问。OOD 中根据具体的编程语言，对严格的封装原则进行必要的调整，如果语言允许直接访问，则只要修改相应的访问控制即可。另一方面，还需考虑 OOD 设计模型的几个外围组成部分而相应增加或修改的一些类、属性和服务。

(3) 设计对象和服务：按照具体的环境和语言的研究，详细设计每个对象类的属性和服务。

在面向对象的分析中我们已经完成了商场销售管理系统的面向对象的分析。在对问题域部分进行设计的时候可以将 OOA 的结果直接带入 OOD 中的 PDC。商场销售管理系统的 PDC 完全展开图如图 6.30 所示。

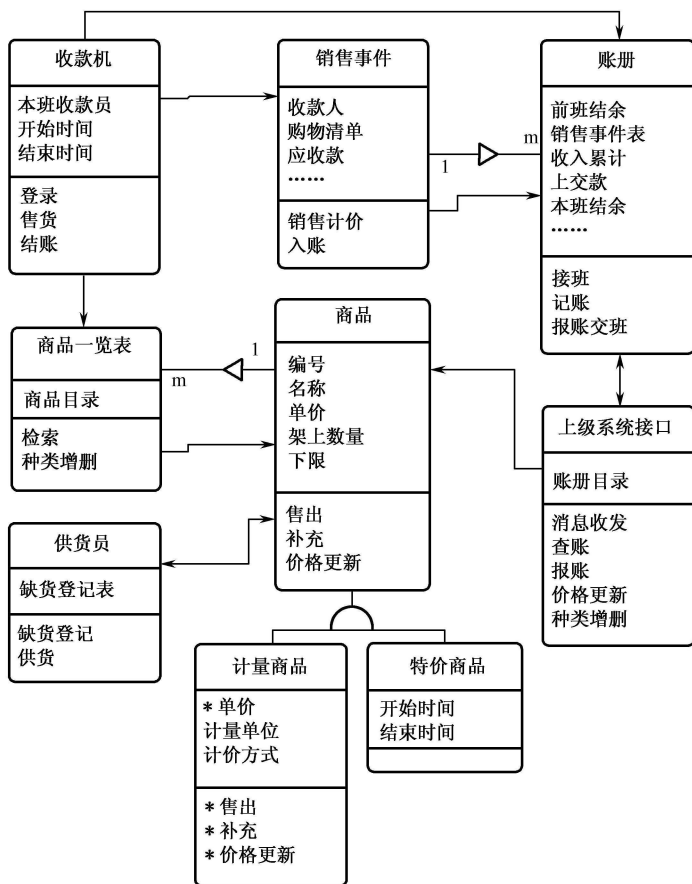


图 6.30 商场销售管理系统的 PDC 完全展开图

### 6.4.2 人机交互部分的设计

人机交互部分是 OOD 外围组成部分之一，它所包含的对象构成了系统的人机界面。现在，计算机的人机界面大多采用图形用户界面系统。它以形象、直观、易学、易用等特点拉近了计算机与用户之间的距离，也是计算机得以迅速为用户所接受的基础。但是图形用户系统的开发工作量也很大，占用的比例很高。目前，使用最多的图形用户系统有 MS-Windows、X Windows、Unix 等。在开发方面目前也有许多成熟的 GUI 图形库帮助开发人员开发一个图形应用系统，如 Visual C++ 的 MFC、OpenGL 等，这些图形用户系统和开发工具可以使开发的效率显著提高，但是不同的用户界面

支持系统，在概念、术语、风格、结构和支持界面开发的程度和级别等方面存在着较大的差异，使得开发的系统中与用户界面相关的对象将受到很大的影响。人机交互部分的设计将最大限度地隔离可能对问题域部分产生的影响，也就是说，当界面支持系统变化时，问题域部分可以保持基本不变。

人机界面部分不是纯粹地设计和实现问题，很大程度上也是在分析问题。在用户需求中可能包括用户对操作的要求，也可能没有包括。这就要求在设计时充分考虑用户的要求，通过讨论协商等方法获取界面要求，并将这些要求反映在设计中。另一方面，人机界面又不完全是一个软件问题，它还需要许多其他的科学知识。它涉及诸如美学、心理学和艺术方面的内容，如人机界面是否美观、布局是否合理等。

人机交互部分突出人如何命令系统及系统如何向用户提交信息，人在使用计算机过程中的感受直接影响到他（她）对系统的接受程度。随着计算机应用的不断普及和深入，非计算机专业人员在使用计算机的人群中所占的比例也在不断增加，所以人机交互部分的友好性直接关系到一个系统的成败。

人机交互部分的设计准则包括以下几个方面。

（1）使用简便：使用者完成一次与系统的交互，所需要的操作应尽可能少，包括使用的键盘、鼠标的次数等；另一方面，界面提供的选择信息（如菜单选项、图标等）也要适量，并排列合理、意义明确。

（2）一致性：界面中的各个部分和层次，在术语、风格、交互方式、操作步骤等方面应尽可能保持一致，并与当前“流行”的设计风格接近。

（3）启发性：能够启发和引导用户正确、有效地进行操作。界面上出现的文字、图形和符号应具有启发寓意，提示信息及时和明确，布局和组织合理。新用户可以不需要或直接从帮助中得到相关的操作指导。

（4）减少记忆负担：使用者在与系统交互时不必记忆大量的操作规则和对话信息。在设计时，应该是给机器（计算机）编程序，而不是给人编程序。假如设计的用户界面需要用户记住大量的使用规则、操作步骤和注意事项，使用者像奴仆一样在系统面前小心翼翼，不知所措，则系统将很难为用户接受。

（5）减少重复输入：系统能够记忆用户已经输入的一些数据，特别是较长的文字数据。在其他功能中，可以直接自动地或通过简单操作复用以往的输入信息，而不需要重新输入。

（6）容错性：对用户操作失误的容忍程度和补救措施，包括对可能引起的不良后果的操作给出的警告信息或再次确认，以及提供操作的撤销（Undo）和重复（Redo）的能力。

（7）及时反馈：对于需要较长时间进行处理的功能，不要等操作完成后才做出反

应。这样使用者不知道自己的操作是否正确或者系统是否存在问题，因此，系统应通过某种方式及时反馈系统中的操作、处理情况，可以采用进度条等方式告诉用户目前已经处理到了何种程度，也可以通过动画方式告诉操作者“处理正在进行中”等。

(8) 其他：其他评价准则包括艺术性、趣味性及风格等方面的因素，这是在界面设计时也必须考虑的重要因素。

设计人机交互部分的策略由以下几点构成：

- 对使用者（人）分类；
- 描述人及其任务脚本；
- 设计命令层；
- 不断原型化；
- 设计人机交互部分的类。

这些活动的次序反映了在人机交互设计中正常的优先事项集合：首先是人，其次是任务，然后是工具。

### 1. 对人分类

设计实现满足用户习惯和需要的操作方式，必须首先研究用户，设身处地地考察用户的实际情况，身临其境地看他们如何实际地完成业务工作。其中需要考虑用户希望系统能够达到什么样的目的？要完成什么样的任务？设计者能提供什么工具来支持这些任务？工具如何做得最协调等。

可以将使用者按照某种分类准则分为不同的类型。例如，考虑按以下的一个或几个原则分类：

(1) 按技能层次分类：

初学者/临时人员/中级水平/高级水平

(2) 按组织层次分类：

行政人员/办公人员/职员/管理人员/办事员

(3) 按不同组的成员身份：

职员/顾客

对于划分为不同类型的使用者，它们各自的操作习惯和使用方式或多或少都有一些区别，例如，对于初学者和临时人员，要求直观、简便并提供有效的使用帮助；对于中级和高级使用者，则应提供快捷、灵活的操作模式。

### 2. 描述人及其任务脚本

对于上面定义的每一类使用者，考虑下述问题并制表：

- ① 谁；
- ② 目的；
- ③ 特征（年龄、教育水平、限制等）；
- ④ 关键的成功因素；
- ⑤ 必须/想要；
- ⑥ 喜欢/不喜欢/有偏见；
- ⑦ 熟练程度；
- ⑧ 任务脚本。

对每一类使用者，把这些问题搞清楚，并写下其中每一个人的情况。这里有运用这种方法的一个例子：

谁：一个分析员

目的：做实际的分析工作。他需要一个图形化的工具能帮助他更有效地工作。

特点如下。

年龄： 岁。

教育水平：大学毕业

关键的成功因素：工具必须使我做有效的分析时畅通无阻。工具不与正在做的工作冲突。

另外，它能在任何时刻给出模型任何部分的文档。

熟练程度：高级熟练程度。

任务脚本：识别“核心”的类及对象，然后识别“核心”结构，发现属性和服务就把想到的东西加进去，到后来的工作中在模型中看到它们。

检验模型：打印模型及其全部文档。

### 3. 设计命令层

设计命令层包括下列工作。

#### （1）研究现有的人机交互活动的寓意和准则

如果命令层将放在一个已建立的交互系统之中，就从研究现有的人机交互活动的寓意和准则开始，对于图形用户界面更是如此。这种准则可能是非正式的（例如，“它看起来及在感觉上就像这样”），也可能是正式规定的（例如，Apple 计算机公司为 Macintosh 制订了相当广泛的准则）。在上述各种情况下，人机交互活动的寓意和准则都是随时间演化的，因此需要继续观察、研究、做原型，并吸取好的例子。

#### （2）建立一个初始的命令层

命令层可能以多种形式呈现给用户：

- 命令行；



- 一个菜单条；
- 一系列图标，当单击它们时会激活一个动作。

命令层是用过程抽象组织可用服务的一种体现。在这一点上，设计转到了如何体现用过程抽象组织所需的服务。

### (3) 细化命令层

为了细化命令层，要考虑排列、整体/部分组合、宽度与深度的对比、减少操作步骤等问题。

① 排列。在开发这个层次时，要细心地选择醒目的服务名，并在这个层次的部分合理安排服务名：把使用最频繁的服务名放在前边；按照用户的工作步骤排列。

② 整体/部分组合。通过服务本身发现整体/部分模式，以帮助在命令层中对服务进行组织和分块。

③ 宽度与深度的对比。工作准则是，不要超过人类短期记忆的局限。Miller 博士在《魔数  $7 \pm 2$ ：我们处理信息的能力局限》一文中指出，人类的短期记忆能力限制在同时能记住  $5 \sim 9$  件东西。Miller 于 1975 年又在《十五年后的魔数 7》一文中又重新考虑了此事，他说  $7 \pm 2$  原则不如改为每次记忆 3 块，每块至多有 3 项。

应用 Miller 的方法，查看命令层中宽度与深度的对比，努力寻求每块有 3 项的 3 个块的宽度，把深度限制在大约 3 层左右，这样用户就不必为知道他进入应用有多远而大伤脑筋了。

④ 减少操作步骤。使点取、推动及键盘操作减少到最少并能完成用户所要求的工作。同时，为系统的高水平用户提供操作捷径。

## 4. 不断原型化

原型化人机交互行为对人机交互部分的设计来说是至关重要的一步，用户需要不断使用、细化假设的交互界面，以进一步完善和改进人机交互部分的设计。

## 5. 设计人机交互部分的类

人机交互部分的类在很大程度上依赖于所选用的图形用户界面，如 X Window、Motif、Windows、Presentation Manager、MacApp 或 Smalltalk，不同的图形用户界面具有基本的区别，如字型、坐标系统及事件等。此时，需要根据选定的图形用户界面及前面的策略设计人机交互部分的类，包括窗口、菜单、滚动条、按钮等。

在商场销售管理系统中主要的工作流程可以描述如下。

对人分类：



① 售货员，接班时登录进入销售系统，按照商品的价格进行计价、收款及打印顾客的购货单；下班时结账，退出系统。

② 管理员，按照其权限登录进入系统，对商品进行增、删、改、指定特价商品；查询销售记录、账目；统计商品的销售情况；填写缺货登记表，通知供货员供货。

设计命令层如下：

系统登录	商品销售	商品维护	系统查询
	计价	增加商品	销售记录
	收款	删除商品	商品数量
	打印购货单	修改商品	账目
	交接班	设定特价商品	统计销售情况
			缺货登记

6.4.3 任务管理部分的设计

任务又称为进程（进程是一连串的活动，由其代码所定义），若干任务并发执行时叫做多任务。下列几类系统是需要多任务的：

- （1）负责局部设备的数据采集及控制的系统；
- （2）某些人机界面——其中的多窗口可被同时选来作输入；
- （3）多用户系统，一个用户任务可能有多份复制品；
- （4）多子系统软件结构，任务可能被用作程序片之间的协作和通信；
- （5）负责与其他系统通信的系统。

单处理机上的多任务，可能需要一个任务在其他任务执行期间与它们协作和通信。在操作系统的支持下，这些任务以时间片轮转的方式运行，产生同时运行的感觉。多处理机硬件结构，每台处理机需要独立的任务，此时要增加支持进程间通信的任务。任务增加了设计、编码和过程的复杂性，因此必须细心地选择并作最终调整。但是对某些应用来说，任务能简化总体设计和编码。独立的任务把必须并发执行的行为分离开来。这种并发行为可以在多个独立的处理机上实现，或者在运行多任务操作系统的单处理机上模拟。

任务的选择和调整，遵照下述策略：

- ① 识别事件驱动任务；
- ② 识别时钟驱动任务；
- ③ 识别优先任务和关键任务；
- ④ 识别协调者；

⑤ 审查每个任务；

⑥ 定义每个任务。

这种策略的要点是识别并设计任务，加上包含在每个任务中的服务。

### 1. 识别事件驱动任务

有些任务是事件驱动的，这些任务可能是负责与设备、其他处理机或其他系统通信的。任务可以设计成由某个事件来触发，该事件常常针对一些数据的到达发出信号。数据可能来自输入数据行或者来自由另一个任务写入的数据缓冲区。当系统运行时，这类任务的工作过程如下：任务睡眠（不消耗处理机时间），等待来自一个数据行或其他来源的中断；当接收到中断后，任务醒来，读取有关数据，把数据放到内存的缓冲区或其他目的地，对所有需要知道此事者发出通知，然后又回到睡眠状态。

### 2. 识别时钟驱动任务

这类任务按特定的时间间隔被触发去进行某些处理。某些设备需要周期性地数据采集和控制，某些人机界面、子系统、任务、处理机或其他系统可能要周期性地通信——这正是时钟驱动任务的用途。

当系统运行时，这类任务的工作过程如下：任务设置一个唤醒时间，然后去睡眠；任务睡眠（不消耗处理机时间），等待来自系统的一个时钟中断；当接收到这个中断后，任务醒来，进行必要的处理，然后又回到睡眠状态。

### 3. 识别优先任务和关键任务

优先任务既包括高优先级的也包括低优先级的处理需要。关键任务用来分离出对于系统的成败特别关键的那些处理，这种处理通常具有特别严格的安全性要求。

#### （1）高优先级

有些服务是高优先级的，这种服务可能需要划到一个独立的任务中，使该服务在一个紧迫的时间约束下完成。可能需要用附加的任务把这种服务分离出来。

#### （2）低优先级

与高优先级恰恰相反，有些任务是低优先级的，可以进行低优先级的处理（常常称作后台处理）。可能需要用附加的任务把这种服务分离出来。

#### （3）高临界性

有些服务对于系统的持续操作可能是至关重要的，而且有些服务甚至在降级的操作方式期间对于持续的操作可能也是必不可少的。可用附加的任务来分离这种关键处理，这种分离隔开了高度安全性处理所需的深入细致的设计、编程和测试。

#### 4. 识别协调者

有三个或更多的任务时，可考虑另外增加一个任务，这个任务起到协调者的作用。这样一个任务加在上边时，现场切换时间（从一个任务转到另一个任务的时间）可能使这种设计方法遇到困难，但这个任务可为封装任务之间的协作带来好处，其行为可以用一个状态转换矩阵描述。用这样的任务来协调其他任务，不要用它去实现那些本应属于分配给被协调的任务所包含的类及对象的服务。

#### 5. 审查每个任务

必须使任务的数目保持到最少额度。

无论在开发阶段还是在维护期间，每次仅仅能理解一个或少数几个正在进行的任务。设计多任务协调的主要问题之一是设计者常常迷恋于定义太多的任务。例如，一个设计者可能仅仅为了在处理他自己的小小的设计与编程问题时寻求方便而增加任务，此时总体设计的可理解性和技术复杂性可能要受到损失。因此要审查每个任务，确保它能满足一个或多个选择任务的工程标准——事件驱动、时钟驱动、优先/关键任务或协调者。

#### 6. 定义每个任务

定义任务包括任务的内容、它是如何协调及如何通信的。

##### （1）任务的内容

首先要说明是什么任务，为任务命名，并简要地说明该任务。为 OOD 部分的每个服务增加一个新的约束——任务名，并为该约束分配一个值。

如果一个服务被分解，交叉在多个任务中，则要修改服务名及其描述，使每个服务能映射到一个任务。对那些与设备、其他任务或其他系统协调和通信的服务，要用协议专用的细节来扩充服务的规格说明。

##### （2）如何协调任务

定义每个任务怎样协调工作：指出它是事件驱动的还是时钟驱动的；对于事件驱动的任务，描述触发该任务的事件；对于时钟驱动的任务，描述在触发之前所经过的时间间隔，同时指出它是一次性的还是重复的时间间隔。

##### （3）如何通信

定义每个任务如何通信：任务从哪里取数据（如输入行、信箱、信号量、缓冲区或约定点）；任务往哪里送数据（如输出行、信箱、信号量、缓冲区或约定点）。

以上几段所讲的内容可以由图 6.31 所示的模板表示。

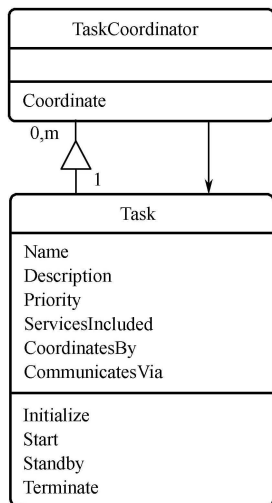


图 6.31 任务管理部分的 OOD 表示模板

在商场销售管理系统中，所有的服务没有并发过程，不需要设计任务管理部分。

#### 6.4.4 数据管理部分的设计

数据管理部分在 OOD 模型中负责与具体的数据管理系统相衔接的组成部分。它将系统中需要长久存储的对象提供了在选择的数据管理系统中进行数据存储的能力。

数据存储问题在大多数系统中都是必须涉及的方面。例如，系统运行过程中的某些数据需要长期进行保存，如录入表格、账簿、统计结果等。凡是需要长期保存的数据，都需要保存在永久存储媒体中，对计算机而言一般是指磁盘（硬盘、软盘、光盘）等。在存储时需要在某种数据管理系统（文件系统或数据库管理系统）的支持下进行数据存储。

在面向对象中，数据是以属性的方式存在于对象之中的，因此数据存储问题表现为对象的存储。这样的对象称为永久对象，即需要长期存储的对象。在进行分析和设计时需要说明那些是永久对象，同时需要设计具体的措施来解决对象的存储问题。数据管理部分就是用面向对象的概念和方法针对具体的存储系统设计相应的解决办法的。

在 OOD 中永久对象的存储可以采用不同的数据管理系统来解决。例如，可以采用文件系统、关系数据库系统（RDBMS），也可以采用面向对象的数据库系统（OOD-BMS）等。从理论上讲，采用 OODBMS 应当是最合适的选择，但是在具体应用系统的开发过程中，有许多需要考虑的因素，这些因素对数据管理系统的选择有很大的影响。有时，选择文件系统或关系数据库系统可能更为适合。

各种数据管理系统都有其各自的优势和特点，但是它们对数据的定义方式和操纵方式却存在较大的差异，例如，不同的文件系统之间存在不同的数据组织方式和命令，不同的数据库管理系统也有不同的逻辑数据模型和操纵语言。设计数据管理部分很大程度上需要根据不同数据管理系统的特点将实现数据格式或数据模型进行转换，并利用它们提供的功能实现数据的存储。显然，针对不同的数据管理系统，需要做出不同的设计。在 OOD 中，根据所选择数据管理系统的特点，设计一些专门用于处理其他对象的永久处理问题的对象，并将它们组织成为一个相对独立的部分，尽可能隔离具体的数据管理系统的差异对对象的影响。也就是说，当选择的数据管理系统发生变化时，只需要数据管理部分的对象发生相应的变化，而系统其他部分的对象基本保持不变。

### 1. 数据管理方法

数据管理方法主要包括两个方面：文件系统和数据库系统。文件系统中可以是局部文件系统也可以是分布式文件系统，数据库系统可以是关系数据库系统也可以是面向对象的数据库系统。这里对它们的概念、原理和技术不作过多的说明，仅从系统设计时不同的数据管理系统对设计的影响程度上进行讨论。

#### (1) 文件系统

文件系统一般是操作系统中的一个组成部分。它采用标准、统一的方法对外部存储器中的文件（数据）进行管理，提供存储、检索、更新、共享和保护功能。在文件系统的支持下，应用程序不必直接操纵物理存储设备而实现数据的管理。

文件在存储空间的组织方法和关系有逻辑结构和物理结构。其中物理结构是文件系统开发者考虑的问题，使用者更加侧重于逻辑结构。常见的文件的逻辑结构有：流结构，即整个文件由顺序排列的字节构成，除此之外再没有其他的结构关系；记录结构，即文件的结构由若干记录构成，每个记录是某个逻辑结构，具有一定的含义。文件系统对文件的组织一般按照树形的结构方式组织，即文件由目录和具体文件构成，目录中可以存放文件。

与数据库相比较，文件系统的特点是廉价（一般不需要购买，由操作系统提供），容易学习和掌握（与文件相关的操作命令一般包括创建、销毁、打开、关闭、读写、共享和控制等），对被存储的数据和类型没有特别的要求。与数据库系统相比，其命令和使用要简单得多，但提供的功能有限。文件系统的局限性如下：

- ① 各个文件中的数据是相互独立和分离的，不易直接体现数据直接的关系；
- ② 容易产生数据冗余，对数据完整性的维护带来困难；
- ③ 应用程序依赖于文件的结构，当文件的结构发生变化时，程序需要做相应的变化；
- ④ 不同程序语言产生的文件之间的差异很大，互不兼容；

⑤ 难以按需要表示数据的视图。当需要表示数据之间的关系（视图）时，难以把文件之间的数据结合成自然的、符合用户需要的数据视图。

## （2）数据库系统

鉴于文件系统的局限性，20 世纪 60 年代中期开始出现数据库技术。经过不断改进和发展，数据库技术已经成为信息系统中的重要技术组成和应用部分，起着重要的作用。数据库就是在计算机中长期存储、组织、管理和共享的数据集合。它按照一定的数据模型来组织、描述和存储数据，具有冗余度小、数据独立性高和易扩展等特性。

数据库是按照一定的数据模型来组织数据的。自从数据库出现以来，先后出现过层次数据模型、网状数据模型、关系数据模型和面向对象的数据模型等。其中，关系数据库是目前应用最为广泛的数据库，面向对象的数据库则是面向对象在数据库领域的扩展和应用。

### ① 关系数据库系统。

关系数据库系统建立在关系理论的基础上，用二维表来表示各种数据。二维表中有行和列。每个列称作属性，每一行称作元组，整个表成为一个关系。一个二维表既可以用来存放实体本身的特征及其属性，也可以存放实体之间的关系。表与表之间定义了操作，如选择、投影及连接等。

表中的每一行都应该是唯一的。其唯一性是将一列或多列定义成表的主关键字（Primary Key）——表中各行的唯一标识。也可定义外来关键字（从其他表定义并获得的關鍵字），以反映表之间的关系，同时提高相关表之间的行访问速度。

关系模式中的“二维表”的构造应减少数据冗余及保证数据一致性。消除数据冗余的方法可用“范式”来定义。第一范式（1NF）的冗余最大，由此类推而减小。

- 第一范式：属性值必须为原子，指的是该属性值仅仅是一个值而不含内部结构。
- 第二范式：首先它必须符合第一范式的条件，并且其中每个非关键字的属性都可由整个关键字来确定（而不是仅仅由关键字中的一部分来确定）。
- 第三范式：符合第二范式的条件，并且各非关键字的属性仅仅与关键字有关且不是其他非关键字的属性的进一步描述。
- BC 范式：符合第三范式的条件，并且两个以上的非关键字的属性不能映射到另一个非关键字的属性中。

范式层次越高，表的数目越多，复杂服务也增加，系统开销也提高。从实际角度来说，选择第三范式比较合适。

### ② 面向对象的数据库系统。

采用面向对象数据模型的数据库称为面向对象的数据库（OODB），相应的数据库系统称为面向对象的数据库系统（OODBMS）。面向对象的数据库是在面向对象的程序设计和开发方面得到成功应用之后，在数据库领域的深入和扩展。

面向对象的数据库必须具有两个特征：首先它必须是面向对象的，即它必须支持对象、类、继承、封装、多态等面向对象的基本概念；其次，它必须是数据库系统应有的特征和功能，即它必须提供数据定义与操纵语言、数据库维护（安全机制、完整性、并发控制、故障恢复等）、事务运行管理等功能。

自 20 世纪 80 年代以来，OODBMS 陆续有产品问世。这些产品大致分为三类：第一类在面向对象的语言的基础上，增加了数据库管理功能，如 GemStone 和 Object-Store；第二类是对关系数据库系统进行扩充，使之支持面向对象的数据模型，在关系数据模型的基础上提供对象管理功能，并向用户提供面向对象的应用程序接口，如 Iris 和 Postgres；第三类是全新的 OODBMS，即按照面向对象的数据模型进行全新设计，而不是在 OOPL 和 RDBMS 上扩充，如 O<sub>2</sub> 和 DAMOKLES。

## 2. 数据管理系统的选择

对于采用面向对象方法分析、设计而建立的系统模型，可以选择不同的数据管理系统来完成模型的数据存储。从理论上来看，采用面向对象的数据库管理系统是最合适的选择，但是在具体的问题域和需求的情况下，应从实际出发，考虑多种因素后做出选择。一般来说，需要考虑非技术因素和技术因素两个方面。

### (1) 非技术因素

在非技术方面，主要考虑项目的成本、工期、风险和宏观计划等方面的问题。这些方面在项目中有时比技术方面更具有决定意义。

① 数据管理系统的成熟程度和先进性：显然二者是矛盾的，保守、稳健、成熟的系统，可以降低失败的风险；先进、未必成熟的系统，风险较大，但是可能获得更好的发展空间。目前，文件系统和 RDBMS 属于比较成熟的产品；OODBMS 比较先进但尚不够成熟。

② 价格：文件系统价格低廉（通常不需要单独购买）；RDBMS 系统的选择余地很大，有适合个人应用需要的，有适合小规模应用的，有适合大规模应用的，可以根据问题域的规模进行选择，其价格差异很大；OODBMS 价格比较昂贵，目前缺乏广泛的应用来验证它的适应性。

③ 开发队伍的技术能力：文件系统的功能比较简单，容易掌握；RDBMS 较为复杂，但普及程度高，容易获得相关的技术人才；OODBMS 系统的普及程度低，掌握它的人少。因此，需根据技术能力的程度进行适当的选择。

④ 其他：与系统相关的其他一些因素。比如，即将开发的系统对整个系统中已经存在的系统的影响。它们往往需要与其他系统交换数据或者进行系统集成，因此选择时必须考虑诸如此类的影响。



## (2) 技术因素

① 文件系统：文件系统无疑是最简单、易用的数据管理系统。它几乎可以存储任何类型的数据（包括复杂结构的数据、图形、图像、音频、视频等），适应范围广。但是它的操作低级，需要精确控制文件中的各项数据，而且不同人员可以设计自己的文件格式，造成文件的种类繁多，互相之间不兼容，而且数据的完整性、共享等方面的功能存在缺陷。

② 关系数据库系统：RDBMS 系统在数据存取、数据共享、完整性、故障恢复和事务处理等方面具有很强的能力。它非常适合需保存、管理大量数据的应用系统的需要，而且数据访问方法已经标准化（SQL 标准），开发接口也基本统一（如 Windows 下的 ODBC、ADO、OLEDB 技术）。尽管它具有很好的优点，但是关系数据模型对数据模式的限制较多，例如，数据库中的表必须至少是第一范式，即表中每个属性必须是原子的，不能有内部结构。而在面向对象中，对象的属性可以是原子的，也可以是结构的和对象的，因此，在采用 RDBMS 存储对象的数据时需要进行转换才能完成。

③ 面向对象的数据库系统：从技术上来看，用面向对象方法开发的系统采用 OODBMS 系统是再合适不过了。显然，系统开发的各个阶段从系统分析、系统设计到系统实现由于采用统一的方法，所以如果数据管理也采用 OODBMS，则整个过程不存在表示的不一致，也不存在技术的差异，这是最好的选择。但是，实际中，OODBMS 本身的技术尚不够成熟，而且具体的 OODBMS 之间存在较大的差异，满足面向对象的数据模型的程度不一，另一方面，用户在 OOA、OOD 和 OOP 过程中建立的模型与 OODBMS 的匹配程度不尽一致，所以也不容小视其对实际工程的影响。

## 3. 数据管理部分的设计

数据管理部分需根据不同的数据管理系统具有针对性地做出，下面简单讨论采用文件系统和关系数据库系统的一些方法和策略。

### (1) 文件系统

如果用文件系统实现对象的存储，那么在 OOA&D 阶段为应用系统识别、定义的对象，在实现时将被表示成文件中的数据。显然，将对象保存成为文件时必须将对象进行转换，对对象的简单属性一般不需要转换，直接存储即可；对于复杂结构和对象属性的属性在存储时比较复杂时，需要单独考虑。

一般来说，为了保持分析、设计、实现的一致性，数据存储尽量不要对它们产生过多的影响，因此，一般需要设计专门的数据接口对象，由这些对象完成数据的存储工作，而其他对象则不需要或者很少改变，如图 6.32 所示。

对于类而言，有多个对象实例，对象实例的存储方式可以有不同的映射方式，如图 6.33 所示。



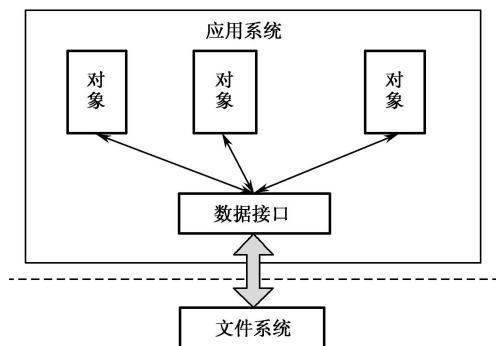


图 6.32 文件系统存储对象

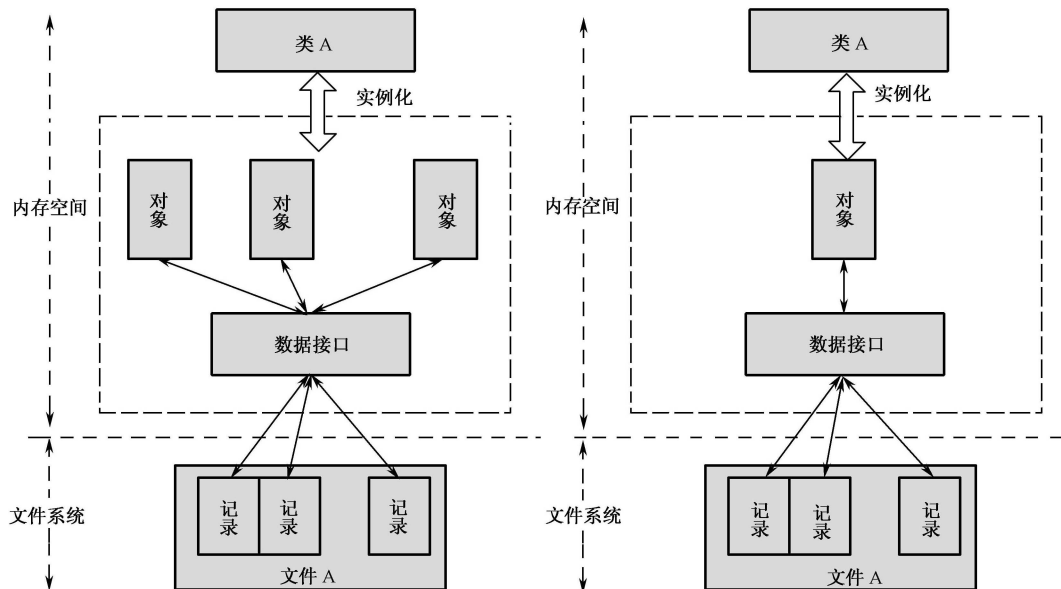


图 6.33 对象实例存储的两种不同映射方式

## (2) 关系数据库系统

与文件系统存储对象的策略类似，关系数据库存储对象的方式是将对象的属性存储在数据库中的二维表中，表中的列存储对象的简单属性，对于对象的复杂属性，可以将其分解为简单属性，并在其他二维表中存储。

显然，需要设计专门用于对象与数据库存储之间的转换对象，负责将数据从数据库中读取出来并赋予相应的对象。这个过程对系统中的对象应该是透明的，即对象仅仅向接口部分提出请求，而不需要知道数据在数据库中是如何具体存储的。在关系数

数据库中对象存储的方式如图 6.34 所示。

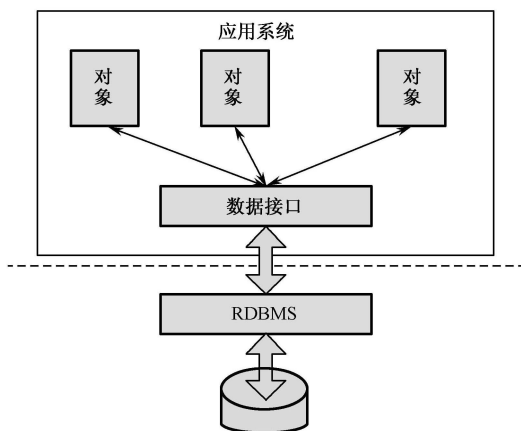


图 6.34 关系数据库系统存储对象

对于类而言，有多个对象实例，对象实例的存储方式可以有不同的映射方式，如图 6.35 所示。

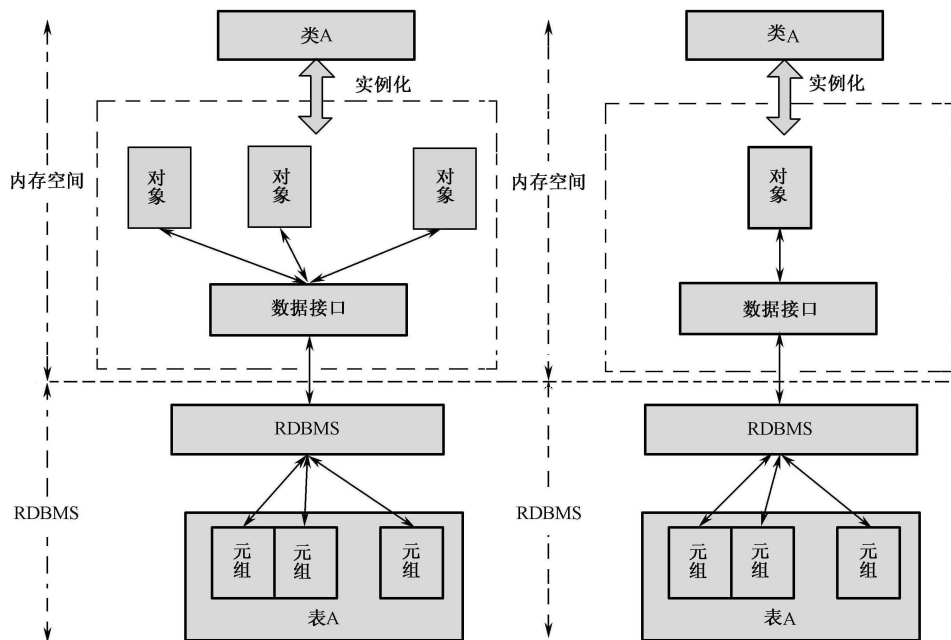


图 6.35 对象与数据库表元组的不同映射方式

### 6.4.5 面向对象设计的评价标准

对于小型系统开发来说，可能只有一个正确的设计方案。而对于大而复杂的系统来说，尤其是当使用了自动化工具来支持设计时，设计方案有可能有多种，必须严格保证所选方案的正确性。此时，必须应用一些评价标准来指导这种设计。

对于面向对象设计应用来说，交互式的、逐步完善的程序开发风格十分流行。当存在下述条件时，这种风格将更为有效：

- (1) 开发环境迅速、友好、可交互；
- (2) 开发人员经验丰富；
- (3) 软件应用问题相对较小，最好是单用户应用。

但是，大多数应用是不满足上述条件的，那么应该如何评价一个设计呢？存在可用的评价标准吗？什么样的设计是一个优秀的设计？

我们简单地对优秀设计下一个定义：优秀设计是一种权衡各种因素、最小化整个生命周期的总系统开销的设计。生命周期内的系统开销涉及分析费用、设计费用、程序设计费用、测试和调试排错费用、硬件费用、运行费用（人、硬件等）及维护费用等。在使用面向对象开发技术的过程中，也存在运行时的性能差、系统开销大及响应时间慢等问题，但这是正常的现象，如同汇编语言取代机器语言，第三代、第四代语言刚刚引入时的情况一样。对于绝大部分系统来说，硬件性能不是关键问题。对于大而复杂的系统来说，软件费用仍是一个问题。在大多数系统中，百分之七八十的软件费用都用于软件维护。因而，优秀设计的一个重要特点是：维护容易。

软件的耦合、内聚及重用是衡量设计好坏的重要标准。

#### 1. 耦合性

“耦合”原本用来描述人与人之间相互依赖的程度。在 OOD 中，耦合表示了 OOD 中各组成部分之间的相互联系程度。耦合是一种重要的评价标准，因为它有助于我们集中考虑这样一个问题：系统中某一部分的变化应该最小限度地影响其他部分。

在 OOD 中，寻找对象之间及类之间的联系。在理想情况下，对某一部分的检查和理解不必涉及其他部分，其原则是：尽量减少耦合。耦合强度可以用组元之间传送的信息的数量和复杂程度来测量。在 OOD 中，有两种情况：由消息连接表示的两个对象之间的交互耦合，一般类与特殊类之间的继承耦合。

##### (1) 交互耦合

应尽量减少交互耦合，其基本原则如下。

① 尽可能减少消息连接的复杂性。一般来说，消息连接中不能有多于三个以上的参数，否则就必须考虑简化。

② 消息连接中三个参数的限定也不是绝对的，不遵循这条原则的设计未必就会失败，但是，许多有实际经验的开发提出：与大多数复杂的消息相连的对象是紧耦合性的，对这种对象的修改一定会产生对其他对象修改的“波动效果”。

③ 另外，在极小化单个消息连接的复杂性的同时，也应该简化所发送的消息的数目及对象所接收的消息的数目。

## (2) 继承耦合

提高继承耦合性是十分需要的。继承就是一般类和特殊类之间耦合的一种形式，OOD 中十分需要继承，类可以按照它所继承的服务和属性与其特殊类耦合。为了获得高耦合，各个特殊类应该明确它是一般类的一个特殊类。因此，如果某个类明确排斥了其一般类中的许多属性，那么它与它的一般类之间就不是强耦合的。更为复杂的情况是特殊类从它的一般类中继承了许多属性，但是完全没有使用它们。对于这两种情况，设计人员都应寻找另外的一种一般—特殊结构，使每个特殊类都继承并使用其一般类的属性和服务，从而更加高度地与之耦合。最重要的关键是要仔细地对类进行命名和组织，使之能反映出哪个是比较一般的，哪个是比较特殊的。

## 2. 内聚性

内聚性表示一组人之间相互关联的程度。在 OOD 中，它表示一组 OOD 组成部分之间的相互关联程度。

### (1) 服务内聚

一个服务应该完成一个并且仅完成一个功能。完成多个功能的服务或者只完成一个功能中的部分的服务，都是不可取的。在 OOD 模块中几乎所有单一功能的服务都很小，一般不会超过 30~50 条语句。语句太多的服务，或者块嵌套太多的服务都应该仔细检查，并尽量避免。

另一种评估内聚的方法是，设计人员用一个词组对服务命名，名称应描述了服务的任务。复合语句、多个动词用“先……”、“后……”、“再……”之类的词组都意味着该服务一定执行了多个功能或一个功能片断。对一个高内聚的服务，通常可以做到用一个包含单个动词和单个直接宾语的简单祈使句来准确地为它取名。

### (2) 类内聚

第二种内聚类型是类内聚。其设计原则是，属性和服务应该是高内聚性的，其中不含无用的属性和服务，而且全都是描述类的对象责任的。

### （3）一般—特殊内聚

第三种内聚关型是一般—特殊内聚。其设计原则是，一般类应该确实描述一般的属性与服务，特殊类应该确实描述特殊的属性与服务，并且仔细斟酌其在结构中的位置。

## 3. 重用性

自从20世纪60年代以来，人们就开始讨论软件重用。所谓重用，就是以前开发过的代码、软件可以在新的软件中再次使用而不需要做大的修改。

### （1）代码重用

人们对“代码重用”谈得最多。代码重用的意思通常是，对库中的某个模块进行子程序调用。但它可以采用下面几种形式。

- 源代码的剪贴。这是一种最初级的重用。
- 源程序级的包含。许多编程语言具有把库中的源代码合并到一个程序中的实用功能。
- 继承。面向对象的程序设计语言具有继承的功能，提供了通过可扩充性而达到重用的机制。
- 对目标码的连接。在大多数编程语言中，编译活动后面紧跟一个“连接”活动，它把全部需要（包括重用）的目标模块收集到一起。
- 运行时引用。在有些环境中，程序与其重用库成分之间的“联编”直到程序执行之前并不占用空间。这在解释执行的环境中实际上是很常见的。

### （2）设计结果重用

设计结果重用指的是重用某程序的设计模型，如重用OOD的模型。使用当前的CASE工具，便可实现这种重用。

### （3）分析结果重用

一种更高的重用级别是描述重用，包括OOA模型或其他一些描述形式的重用。这种重用尤其适用于用户需求未变，而计算机系统内、外结构（硬件技术已变）发生变化的情况。例如，一个项目组若需要将一个系统从老的硬件技术支持转化到更新、更强的技术，则适合采用这个级别上的重用。

### （4）重用的组织方法

为了在项目组织中重用软件，最重要的要素就是管理。管理指的是三方面的内容：

- 奖励使用重用软件机制的设计人员，引起大家对重用的重视；
- 专门的领导以激励重用；
- 在组织中创建一个专门的小组，专门负责重用工作。

#### 4. 其他评价标准

除主要的耦合、内聚及重用标准外，还可使用大量的附加标准来衡量设计的质量，如下所述。

##### (1) 设计的清晰度

由于 OOD 中强调重用性，所以清晰的要求就更显重要。当然，人们不会重用那些不理解的设计，设计清晰与否，与下面几个方面有关：

- 用词一致；
- 遵循已有的协议或行为；
- 避免太多的“消息模块”；
- 避免模糊性的类定义。

##### (2) 一般—特殊结构的深度

对于中型系统说来，一般—特殊结构在  $7 \pm 2$  层左右。

##### (3) 保持对象和类尽量简单

设计方法中很强调简明性。保持对象及类的简明性，可以从下面几个方面考虑：

- 避免多余的属性；
- 强调实质内容，避免出现二义性；
- 尽量避免对象之间的协同；
- 一个类中避免有太多的服务。

##### (4) 保持简单的协议

消息协议中的词汇尽量保持简单，如果消息中参数的数目超出平均数目，则可能其中存在错误。另外，如果在协议中出现有关计算机的术语，则意味着该类中某些内容可能不属问题域，而与实现有关，应避免出现这种情况。

##### (5) 保持简单的服务

OOD 中服务一般相当小，甚至不足五行源程序。当然，服务语句的多少与具体实现的语言有关，但如果服务语句超过 200 个语句，则一定不允许。

##### (6) 评价对设计的变动

评价设计质量的另一条标准是：设计维持不变的时间。如果因为终端用户需求的变化，或现有设计的错误、不足而必须修改设计，那么修改的范围有多大呢？理论上来说，设计的变动曲线如图 6.36 所示。在设计初始阶段，设计的变动较频繁。图中，“峰值”意味着出现了系统的设计错误脉冲，或出现了不期望的变动，峰值越高，意味着设计质量越差，重用性也差。

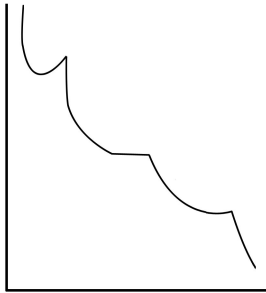


图 6.36 设计的变动曲线

(7) 极小化整个系统的大小

OOD 项目常常是由多人或多个设计小组共同完成的。对于 6 人左右的设计小组来说，一个系统大约包括不超出 100 个类就可以了。但对于具有好几百个类的系统说来，涉及人员太多，需要形式化的 OOD 表示、文档及评估标准。系统太大，不利于清晰地设计，因此，应尽量减小系统的大小。

(8) 脚本评估

还有一种评估设计的方法，即脚本评估法。使用脚本来测试设计的正确性，来寻找完善该设计的方式。在面向对象方法中，设计人员很容易找到可模拟的对象，对象既有状态也有行为。人们将需要模拟的事物假想为对象，然后根据设计中的细节来模拟对象的行为。

(9) 关键成功因素的评价

OOD 中典型的关键成功因素包括可重用性、可读性和性能。

需要为用户提供一部分已实现的设计，然后从各方面来评价各个关键成功因素。注意，必须从多个设计角度（最好有合适的 CASE 工具支持）来评价关键成功因素。

6.5 统一建模语言（UML）

统一建模语言（Unified Modeling Language，UML）是一个在多种面向对象的建模方法联合的基础上形成的统一建模语言，并于 1997 年被对象管理组织（Object Management Group，OMG）采纳为建模语言规范。它是一种通用（General）的建模语言，具有创建系统的静态结构和动态行为等多种结构（Construction）模型的能力。统一建模语言的应用领域很广泛，可用于信息系统需求分析、设计和开发的各个阶段。

### 6.5.1 UML 的发展过程

由于面向对象的分析与设计 (OOA&D) 方法的重要性日益突出, 所以人们对它的研究、开发和应用的热情也在不断升高。在 1989 年, 以专著、论文或技术报告等形式提出的 OOA&D 方法或 OO 建模语言有近 50 种, 到 1994 年, 其数量增加到 50 种以上。各种方法的出现都对 OOA&D 技术的研究与发展做出了或多或少的贡献。这种“百花齐放”的繁荣局面表明, 面向对象的方法与技术已得到广泛的认可并成为主流。然而多种方法的同时存在也带来了一些问题: 各种 OOA&D 方法所采用的概念既有共同部分也有一定的差异 (例如, 许多方法在 OO 基本概念的基础上各自提出了一些扩充概念。字面上相同的概念其语义解释也不尽相同); 在表示符号上, OOA&D 模型及文档组织等方面的差别则更为明显。这种情况常使一些新用户在进行建模方法及工具的选择时感到困难, 也不利于彼此之间的技术交流。

鉴于这种情况, 1994 年同在美国 Rational 软件公司工作的 G.Booch 和 J.Rumbaugh 认为, 应该把他们各自提出的方法 (Booch 方法和 OMT) 结合起来, 形成一种统一的方法。同年 10 月他们开始工作, 并于 1995 年 10 月公开发布了第一个版本, 即 Unified Method 0.8。1995 年秋 OOSE 的提出者 I.Jacobson 加入 Rational 软件公司, 于是也加入了这一行列。G.Booch、J.Rumbaugh 和 I.Jacobson 共同认为, 提出一种统一的建模语言有以下三个理由:

第一, 他们各自提出的方法在演化中已经有互相结合的趋向;

第二, 走向统一将带来市场方面的好处;

第三, 有助于改进他们各自的方法, 以获得更多的学习者, 并解决一些以往他们各自的方法中不能很好解决的问题。

三个合作者在做了认为最好的权衡下, 于 1996 年 6 月和 10 月先后发布了二义性较少的 UML 0.9 和 UML 0.91 版本, 并且将“统一方法”(Unified Method) 改为“统一建模语言”(Unified Modeling Language)。1996 年, Rational 公司准备向对象管理组织 (OMG) 申请将 UML 作为一种标准建模语言, 并为此创立了 UML 伙伴组织, 共有 12 家公司加入, 并推出了 UML 1.0 规范, 于 1997 年 1 月提交给 OMG 组织。之后 UML 伙伴组织又加入了其他一些公司, 并把它们的修改意见反映在 UML 中, 于 1997 年 9 月产生了 UML1.1 版本, 并将其作为最终提案提交给 OMG 组织, 该提案于 1997 年 11 月被 OMG 组织正式采纳。

此后不久, OMG 组织批准成立了 UML 修订任务组 (UML RTF) 收集相关意见和建议, 并负责澄清模糊性和修改建议。UML1.2 修改了 UML1.1 中与 OMG 的其他



规范不一致的地方,并改正了印刷错误、语法错误和某些明显的逻辑上不一致的地方。之后又进行进一步修改而成为 UML1.3 版本,并于 1999 年 6 月提交给 OMG 组织,获得通过(目前广泛流行的就是这个版本)。2001 年 2 月提交形成了 UML1.4 版本。2007 年 6 月发布了 UML2.0 版本。UML 的发展过程如图 6.37 所示。

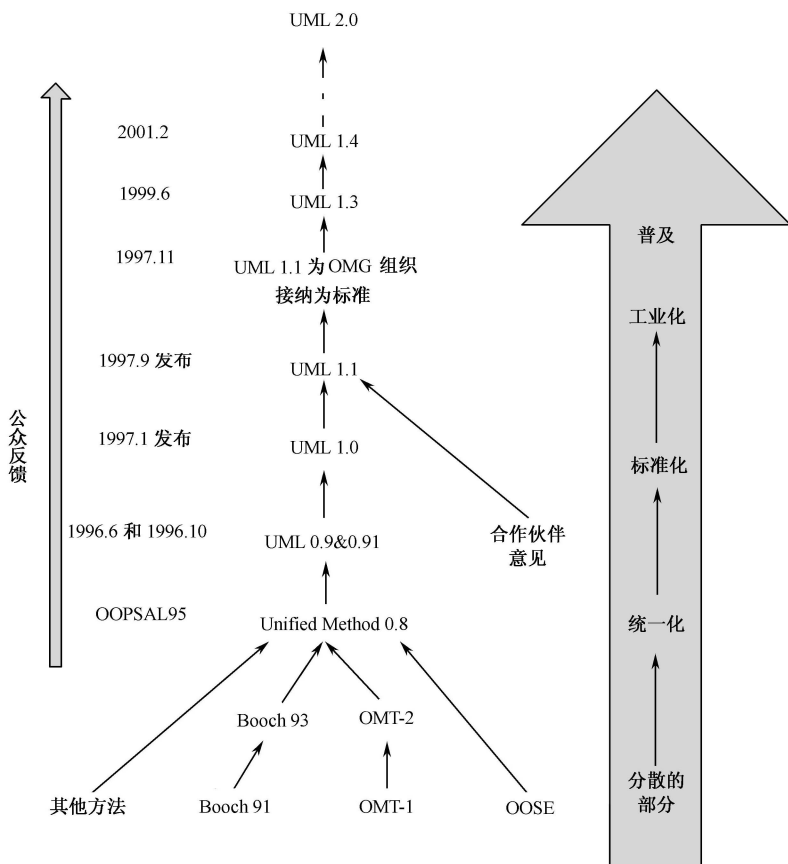


图 6.37 UML 的发展过程

### 6.5.2 UML 的特点

统一建模语言 UML 的主要特点可以归结为以下几点。

(1) 统一了 Booch、OMT 和 OOSE 等面向对象的方法中的基本概念。在 UML 出现以前,出现了很多 OO 方法,这些方法中基本概念的术语、内涵和外延多少都有一

些差异。UML 的出现统一了面向对象的术语及其含义，为面向对象的技术和方法进一步深入发展打下了基础。

(2) UML 吸取了面向对象技术领域其他流派的长处，其中也包括非 OO 方法的影响。UML 在开始时就结合了 Booch 和 OMT 方法，之后又增加了 OOSE 方法，在创立了 UML 伙伴组织后更加吸取了各个方面的优点，使得 UML 原本适用于软件分析设计的面向对象的方法扩展为适合于许多非软件领域，如机械系统、企业机构或业务过程，以及处理复杂数据的信息系统、具有实时要求的工业系统或工业过程等。

(3) UML 的建模手段比较全面，具有丰富的表达能力。它提供了用例、逻辑、组件、并发和配置等视图，并有多种图用来描述问题域。

(4) UML 在演变过程中还提出了一些新的概念。例如，新加了版类 (Stereotypes)、责任 (Responsibilities)、扩展机制 (Extensibility mechanisms)、线程 (Threads)、过程 (Processes)、分布式 (Distribution)、并发 (Concurrency)、模式 (Patterns)、合作 (Collaborations)、活动图 (Activity diagram) 等新概念，并清晰地区分了类型 (Type)、类 (Class) 和实例 (Instance)、细化 (Refinement)、接口 (Interfaces)、组件 (Components) 等概念。

(5) UML 具有实用和工业化的特点。UML 首先在 Rational 公司出现，并有多家公司的大力支持，因而其实用性是显而易见的。随着 UML 的不断完善和发展，UML 的普及非常迅速，各种支持 UML 建模的软件工具和学习书籍大量出现，使得 UML 已经成为一个事实上的工业标准。

诚然，UML 的出现和广泛应用为信息系统的开发及许多行业部门带来了巨大的作用。但是，任何事务都存在两面性，同样，UML 也有它的不足，其不足体现在以下几个方面。

(1) UML 是一种建模语言而不是一种方法，即它不是提供如何运用面向对象的概念和原则进行系统建模，而是定义了用于建模的一些元素，以及用这些元素如何构成各种图的规则。它并不能为信息系统开发的工程技术人员提供一套可遵循的原则。

(2) UML 过于庞大、复杂。随着 UML 规范的不完善，它包含的内容也大大增加了。面对 UML 当中众多的图形、符号和规则，用户很难全面、熟练地掌握，大多数用户也仅仅使用其中的一小部分。

(3) 有很多概念使用户感到困惑，含义不清，并很少使用。

经过人们广泛和深入的研究，UML 中存在的问题还包括四元模型体系中的问题、形式化方面的问题、类图 and 对象图并存的问题及协作图方面存在的问题。有关 UML 存在的缺陷和问题，可参考相关文献和 UML 官方网站中的内容。

### 6.5.3 UML 的模型元素和构成

UML 由视图（Views）、图（Diagrams）、模型元素（Model elements）和通用机制（General Mechanism）等几个部分构成。

（1）视图，用来表示被建模系统的各个方面（从不同的目的出发建立，为系统建立多个模型，这些模型都反映同一个系统，且具有一致性）。视图由多个图（Diagrams）构成，它不是一个图片（Graph），而是在某一个抽象层上，对系统的抽象表示。如果要为系统建立一个完整的模型图，只须定义一定数量的视图即可，每个视图表示系统的一个特殊方面。另外，视图还把建模语言和系统开发时选择的方法或过程连接起来。

（2）图，由各种图片（Graph）构成，用来描述一个视图的内容。UML 语言定义了 9 种不同的图的类型，把它们有机地结合起来就可以描述系统的所有视图。

（3）模型元素，代表面向对象中的类、对象、消息和关系等概念，是构成图的最基本的常用概念。一个模型元素可以用在多个不同的图中，无论怎样使用，它总是具有相同的含义和相同的符号表示。

（4）通用机制，用于表示其他信息，如注释、模型元素的语义等。另外，它还提供扩展机制，使 UML 语言能够适应一个特殊的方法（或过程），或者扩充至一个组织或用户。

UML 的主要内容包括以下五类图。

（1）第一类是用例图（Use-Case Diagram），用例图用于显示若干角色（Actor），以及这些角色与系统提供的用例之间的连接关系。用例图仅仅从角色（触发系统功能的用户等）使用系统的角度描述系统中的信息，也就是站在系统外部察看系统功能，它并不描述系统内部对该功能的具体操作方式。用例图定义系统的功能需求，用于需求分析。

（2）第二类是静态图（Static Diagram），包括类图、对象图和包图。其中类图描述系统中类的静态结构，不仅定义系统中的类，表示类之间的联系（如关联、依赖、聚合等），而且也包括类的内部结构（类的属性和操作）。对象图是类图的实例，使用与类图基本相同的标识，其不同点在于对象图显示类的多个对象实例，而不是实际的类。一个对象图是类图的一个实例。包（Package）由包或类组成，表示包与包之间的关系。包图用于描述系统的分层结构，与前面讲到的 OOA 中主题的概念类似。

（3）第三类是行为图（Behavior Diagram），描述系统的动态模型和组成对象间的交互关系。其中状态图描述类的对象所有可能的状态及事件发生时状态的转移条件。

通常，状态图是对类图的补充。而活动图描述满足用例要求所要进行的活动及活动间的约束关系，有利于识别并进行活动。

(4) 第四类是交互图 (Interactive Diagram)，描述对象间的交互关系。其中顺序图显示对象之间的动态合作关系，它强调对象之间消息发送的顺序，同时显示对象之间的交互；合作图描述对象间的协作关系，显示对象间的动态合作关系。除显示信息交换外，合作图还显示对象及它们之间的关系。顺序图和合作图合称为交互图。

(5) 第五类是实现图 (Implementation Diagram)。其中构件图描述代码部件的物理结构及各部件之间的依赖关系。部件图有助于分析和理解部件之间的相互影响程度。配置图定义系统中软、硬件的物理体系结构。

### 6.5.4 UML 建模的一般方法

给复杂的系统建模是一件困难和耗时的事情。这就好比绘画中的写生，无论用多少时间，画多少张画，也很难将实际的景色一点不漏地反映出来。但是，如果从若干个角度出发，将景色一一临摹下来，这样多张画累积在一起就能大致准确地反映实际的景色，从而能基本达到目的。与此道理类似，完整地描述系统，通常的做法是用一组视图反映系统的各个方面，每个视图代表完整系统描述中的一个抽象，显示这个系统中的一个特定的方面。每个视图可由一组图构成，图中包含了强调系统中某一方面的信息。视图中的图应该简单，易于交流，且与其他的图（图用图形符号表示，图形符号代表系统中的模型元素）和视图有关联关系。

UML 中的视图包括用例视图 (Use-Case View)、逻辑视图 (Logical View)、组件视图 (Component View)、并发视图 (Concurrency View)、部署视图 (Deployment View) 等五种。因此，UML 建模的一般方法是绘制多种视图。

#### 1. 用例视图

用例视图 (Use-Case View) 用于描述系统应该具有的功能集。它是从系统外部用户的角度出发，对系统的抽象表示。用例视图所描述的系统功能依靠于外部用户或另一个系统触发激活，为用户或另一个系统提供服务，实现用户或另一个系统与系统的交互。系统的目标是实现用例视图中描述的功能。用例视图中可以包含若干个用例 (Use-Case)。用例用来表示系统能够提供的功能，一个用例是功能请求的一个通用描述。

用例视图是其他视图的核心和基础。其他视图的构造和发展依赖于用例视图中所描述的内容。这是因为系统的最终目标是提供用例视图中描述的功能，同时附带一些

非功能性的性质，因此用例视图影响着所有其他的视图。

用例视图主要为用户、设计人员、开发人员和测试人员而设置。用例视图静态地描述了系统功能，为了动态地观察系统功能，偶尔也用活动图（Activity Diagram）描述作为补充。

## 2. 逻辑视图

用例视图只考虑系统应提供什么样的功能，对这些功能的内部运作情况则不予考虑，为了揭示系统内部的设计和协作状况，需要使用逻辑视图来描述系统。

逻辑视图（Logical View）用来显示系统内部的功能是怎样的，它利用系统的静态结构和动态行为来刻画。静态结构描述类、对象和它们之间的关系等。动态行为主要描述对象之间的动态协作，当对象之间彼此发送消息给给定的服务时产生动态协作。一致性（Persistence）和并发性（Concurrency）等性质，以及接口和类的内部结构都要在逻辑视图中定义。

静态结构在类图 and 对象图中描述，动态协作用状态图、序列图、协作图和活动图描述。

## 3. 组件视图

组件视图（Component View）用来显示代码组件的组织方式。它描述了实现模块（Implementation Module）和它们之间的依赖关系。

组件视图由组件图构成。组件是代码模块，不同类型的代码模块形成不同的组件，组件按照一定的结构和依赖关系呈现。组件的附加信息（如为组件分配资源）或其他管理信息（如进展工作的进展报告）也可以加入到组件视图中。组件视图主要供开发者使用。

## 4. 并发视图

并发视图（Concurrency View）用来显示系统的并发工作状况。并发视图将系统划分为进程和处理机方式，通过划分引入并发机制，利用并发高效地使用资源、并行执行和处理异步事件。除了划分系统为并发执行的控制线程外，并发视图还必须处理通信和这些线程之间的同步问题。并发视图所描述的方面属于系统中的非功能性质方面。

并发视图供系统开发者和集成者（Integrator）使用。它由动态图（状态图、序列图、协作图、活动图）和执行图（组件图、展开图）构成。

## 5. 部署视图

部署视图（Deployment View），也译作配置视图，用来显示系统的物理架构，即

系统的物理部署情况，如计算机和设备及它们之间的连接方式，其中计算机和设备称为节点（Node）。部署视图还包括映射，该映射显示在物理架构中组件是怎样展开的。例如，在每台独立的计算机上，哪一个程序或对象在运行。展开视图主要提供给开发者、集成者和测试者使用。

### 6.5.5 UML 一般建模过程

UML 是一种建模语言而不是方法，这是因为 UML 中没有过程的概念，而过程正是方法的一个重要组成部分。UML 本身独立于过程，这意味着用户在使用 UML 进行建模时，可以选用任何适合的过程。过程的选用与软件开发过程的不同因素有关，如所开发软件的种类（如实时系统、信息系统和桌面产品）、开发组织的规模（如单人开发、小组开发和团队开发）等。用户将根据不同的需要选用不同的过程。然而，使用 UML 建模仍然有着大致统一的过程框架，该框架包含了 UML 建模过程中的共同要素，同时又为用户选用与其所开发的工程相适合的建模技术提供了很大的自由度。

#### 1. 建模过程

用 UML 语言建造系统模型的时候，在系统开发的每个阶段都要建造不同的模型，建造这些模型的功能和目的也不同。需求分析阶段建造的模型用来捕获系统的需求、描绘与真实世界相应的基本类和协作关系。设计阶段的模型是分析模型的扩充，为实现阶段作指导性的、技术上的解决方案。实现阶段的模型是真正的源代码（Source Code），编译后的源代码就变成了程序。最后是部署模型，它在物理构架上解释系统是如何展开的。

虽然这些模型各不相同，但通常情况下，后期的模型都由前期的模型扩展而来，它们在语义的表述上基本相同。因此，每个阶段建造的模型都要保存下来，以便进行修正或重新扩展、分析模型，如图 6.38 所示。

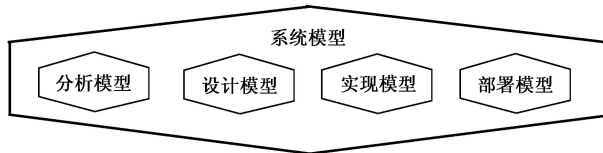


图 6.38 用多个模型描述的系统

UML 语言具有阶段独立性，也就是说，同样的通用语言和同样的图可以用在不同的阶段为不同的事物建模。使用 UML 语言建模，建模工作一定要依照某个方法或过程

进行。建模的过程一般被分为以下几个连续的重复迭代阶段：需求分析阶段、设计阶段、实现阶段和部署阶段。

最后，在实际解决问题的时候，模型被实现为各种原型。生成原型时，要对生成的原型进行评价，以便发现可能潜在的错误、遗漏的功能和开发代价过高等不足之处。如果发现了上述不足之处，那么开发人员还要返回到前期的各个阶段步骤，排除这些问题。如果问题很严重，则开发者或许最终要返回到开始时的集体讨论、描绘草图的阶段，重新建模。当然，如果问题很小，则开发者只须改变模型的一部分组织和规格说明。注意，把图原型化的工作，一定要在把多个图结合成原型结构之后。实际建模工作的大致流程如图 6.39 所示。

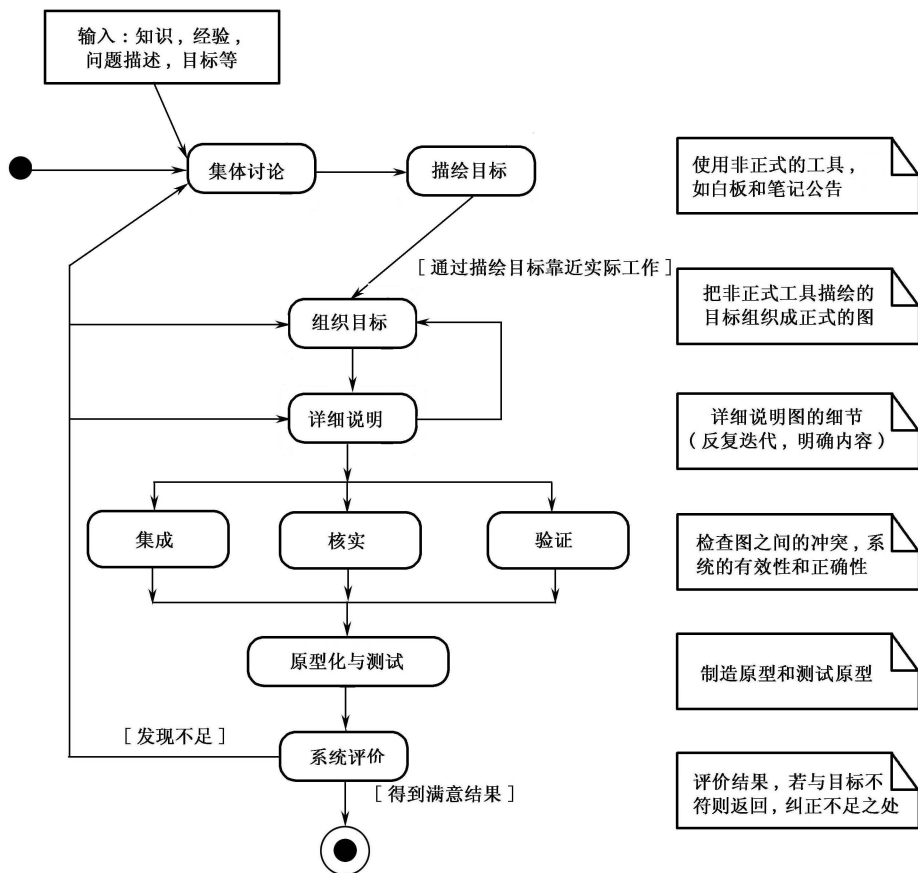


图 6.39 实际建模工作的大致流程



原型只是一个很粗浅的东西，构建原型仅仅是为了对其进行评价，发现其中的不足之处，而对原型进行进一步的开发过程才算得上真正的系统开发过程。

## 2. 工具的支持

使用建模语言需要相应的工具支持。即使人工在白板上画好了模型的草图，建模者也需要使用工具。因为模型中很多图的维护、同步和一致性检查等工作，人工做起来几乎是不可能的。

自从用于产生程序的第一个可视化软件问世以来，建模工具（又叫 CASE 工具）一直不很成熟。许多 CASE 工具几乎和画图工具一样，仅提供了建模语言和很少的一致性检查，增加了一些方法的知识。经过人们不断地改进，今天的 CASE 工具正在接近图的原始视觉效果，如 Rational Rose 工具，就是一种比较现代的建模工具。但是还有一些工具仍然比较粗糙，如一般软件中很好用的“剪切”和“粘帖”功能在这些工具中尚未实现。另外，每种工具都有属于自己的建模语言，或至少有自己的语言定义，这也限制了这些工具的发展。随着统一建模语言 UML 的发布，工具制造者现在可能会花较多的时间来提高工具质量，减少定义新的方法和语言所花费的时间。

一个现代的 CASE 工具应提供下述的功能。

① 画图 (Draw Diagrams)。CASE 工具中必须提供方便作图和为图着色的功能，也必须具有智能，能够理解图的目的，知道简单的语义和规则。这样的功能带来的方便是，当建模者不适当地或错误地使用模型元素时，工具能自动警告或禁止其操作。

② 积累 (Repository)。CASE 工具中必须提供普通的积累功能，以便系统能够把收集到的模型信息存储下来。如果在某个图中改变了某个类的名称，那么这种变化必须及时地反射到使用该类的所有其他图中。

③ 导航 (Navigation)。CASE 工具应该支持易于在模型元素之间导航的功能。也就是说，使建模者能够容易地一个图到另一个图地跟踪模型元素或扩充对模型元素的描述。

④ 多用户支持。CASE 工具提供该功能使多个用户可以在一个模型上工作，但彼此之间没有干扰。

⑤ 产生代码 (Generate Code)。一个高级的 CASE 工具一定要有产生代码的功能。该功能可以把模型中的所有信息翻译成代码框架，把该框架作为实现阶段的基础。

⑥ 逆转 (Reverse)。一个高级的 CASE 工具一定要有阅读现成代码并依代码产生模型的能力，即模型可由代码生成。它与产生代码是互逆的两个过程。对开发者来说，他可以用建模工具或编程两种方法建模。

⑦ 集成 (Integrate)。CASE 工具一定要能与其他工具集成，即与开发环境（如编



辑器、编译器和调试器）和企业工具（如配置管理和版本控制系统）等集成。

⑧ 覆盖模型的所有抽象层。CASE 工具应该能够容易地从对系统的最上层的抽象描述向下导航至最低的代码层。这样，若需要获得类中一个具体操作的代码，则只要在图中单击这个操作的名字即可。

⑨ 模型互换。模型或来自某个模型的个别的图应该能够从一个工具输出，然后再输入到另一个工具。就像 Java 代码可在一个工具中产生，而后用在另一个工具中一样。模型互换功能也应该支持用明确定义的语言描述的模型之间的互换（输出/输入）。

### 3. 模型积累

CASE 工具的模型积累就是提供一个数据库。用数据库保存模型中元素的所有信息，而不考虑这些信息来自哪个图。这个积累应该含有整个模型的基本信息，这些基本信息可以通过若干个图看到，如图 6.40 所示。

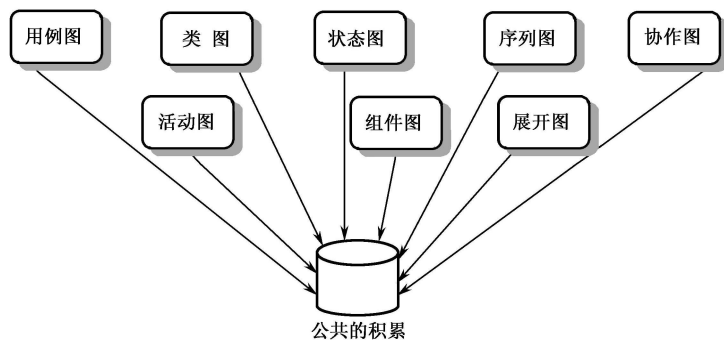


图 6.40 积累包含所有图的信息

在积累的帮助下，CASE 工具的检索不一致、鉴定、报告、重用元素或图的工作才能执行。

① 检索不一致的含义是，如果元素与其在其他图中的用法不一致，则 CASE 工具必须警告或阻止这种用法。如果建模者打算删除某图中的元素，而该元素又被其他图使用，同样也会产生警告开发者的信息。如果开发者一定要删除这个元素，那么所有引用了该元素的图中的该元素也将被删除。这样开发者就一定要返回去修改引用了该元素的图，使之有效。

② 鉴定，是指 CASE 工具可以使用积累鉴定模型，或者指出尚未详细说明的部分，或者跟踪模型，发现可能的错误或不合适的解决方法。

③ 报告，指的是 CASE 工具能自动产生所有模型元素的完全的和扩展的文档，如

类或图的报告。这个报告与所有数据的术语分类很相似。

④ 重用元素或图，指的是利用积累支持重用功能，以便某个工程的建模解决方案或部分方案能容易地用在另一个工程中。UML 模型中的组件直接与源代码相连，因此组件或其源代码都能在不同的工程中重用。

#### 4. 集成

建模工具与系统开发时需要使用的其他工具形成一个整体，即集成。建模工具只是集成环境中的一个部分，但是对其他工具而言，它是一个真正的自然的“集线器”(Hub)，如图 6.41 所示。可以集成的工具有开发环境、配置和版本控制工具、文档工具、测试工具、GUI 构造器工具、需求说明工具和过程支持工具等七种。

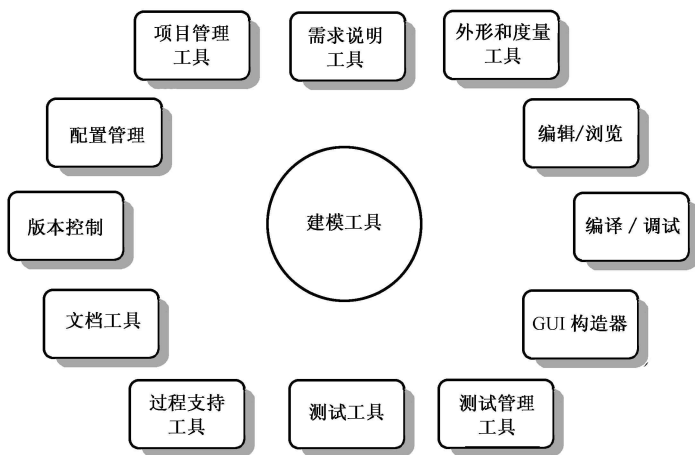


图 6.41 以建模工具为核心集成的工具箱

(1) 开发环境，包括代码的编辑、编译和调试。有了这些集成的工具，就可以从图的模型元素直接进入开发环境，反过来也一样。同样，也能在编辑代码模块时，决定该模块在系统的逻辑和物理模型中的位置。

(2) 配置和版本控制工具，用来控制系统的各种不同的配置，以及系统和个别元素的不同版本。版本控制包含对模型和代码两方面的控制。

(3) 文档工具，能自动从模型积累中产生文档。它还能从已知的信息中产生数据资料。

(4) 测试工具，主要用于管理测试过程，收集和维持测试报告。测试保证系统的有效性（系统好不好？）和正确性（系统对不对？）。模型本身所含有的供测试过程用的大量信息也应该传给测试工具。

(5) GUI 构造器工具，存储和管理图形化的用户接口，它应能自动地为模型中的类生成 GUI 表单 (Form)。比如，通过获取类的属性，自动生成表单中与之相应类型的域。

(6) 需求说明工具，用于对于系统中非功能性方面需求的描述，如定时需求、可靠性和展开性等 (UML 中通过用例捕获系统的功能性需求)。

(7) 设计工程管理工具的目的在于帮助工程管理者制定时间安排表、资源分配计划表和跟踪工程进度表。由于建造模型占据工程的大部分时间，因此使工程管理者能够容易地检查建模工作的进度是很有意义的。

(8) 过程支持工具，用于支持某个具体方法或过程的用法。在 CASE 工具中集成过程支持工具可能不是最佳的选择，因为从本质上讲，建造的模型应是对任何方法或过程实质性部分的体现，而过程支持工具提供的用法不一定适合该方法或过程。

最后需要说明，并不是每个工程都要使用上述所有工具，目前市售的建模工具也没有完全集成上述所有工具。

## 6.6 用例驱动的结构建模和行为建模

在进行实际的信息系统开发时，从 UML 建模到代码实现，一定要依照某个方法或过程进行。从建模规模的两个极端来看，典型的方法既包括统一建模过程 (Rational Unified Process, RUP)，又包括极限编程 (Extreme Programming, EP)。本节介绍基于过程的 UML 建模方法论，将介绍目前在工程实践中所采取的折中方案，如 ICONIX 过程。

### 6.6.1 基于过程的 UML 建模方法论

ICONIX 过程创始于 1992 年，先于 UML 出现，又在 UML 后得到进一步的发展。其规模介于重量级 RUP 和轻量级的极限编程之间 (XP)。而且这种方法也是用例驱动，但不需要像 RUP 那样将需求分析的记录延续到表中而带来大量的开销。它与 XP 类似，相对较小，但不像 XP 那样摒弃了分析和设计过程。因此，有利于在过程中使用 UML，同时对需求进行跟踪。该过程侧重于系统分析和设计，遵循 Ivar Jacobson 的用例驱动思想，能够获得形象、具体、易于理解的用例，开发小组可以使用这些用例来驱动开发工作。ICONIX 过程通过采用“健壮性分析”（也称鲁棒方法），解决了从用例到时序图的鸿沟，这也成为该方法独有的特色，如图 6.42 所示。

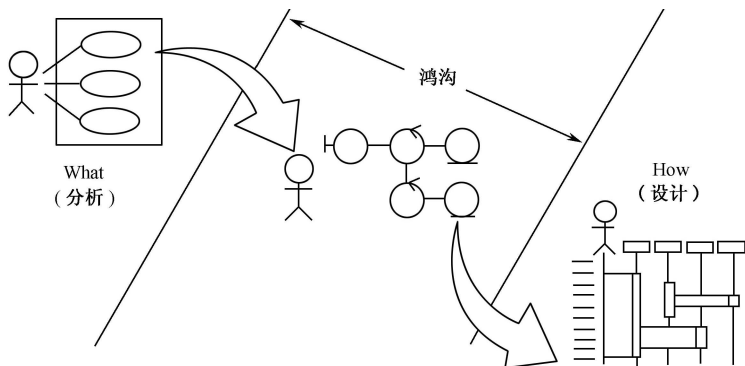


图 6.42 健壮性分析的作用

同时，ICONIX 过程是简化了的、核心的 RUP 建模方法。通过采用迭代的开发方法，循序渐进同时足够的简洁。因为它遵循 20% 原则，即用 UML 中 20% 的图表来满足设计中 80% 的需求，有利于快速开发。

一般地，ICONIX 过程包括以下几个主要阶段，如图 6.43 所示。

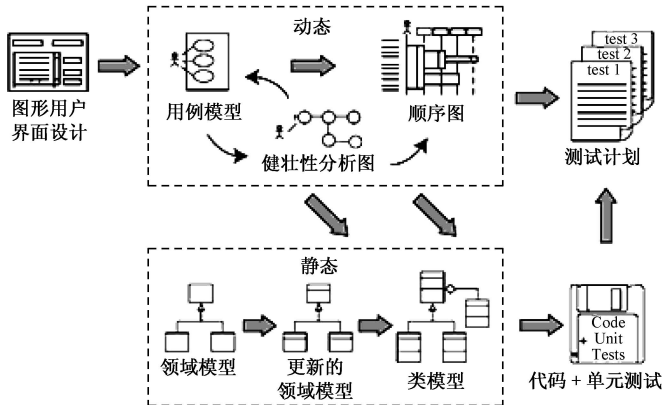


图 6.43 ICONIX 过程示意图

- ① (问题) 域建模 (Domain Model)。
- ② 用例建模 (User Case Model)。
- ③ 需求复核。
- ④ 健壮性分析 (Robustness)。
- ⑤ 初步设计复核。

⑥ 时序图（Sequence Diagram，绘出设计阶段的类图）。

⑦ 关键设计复核。

以下对上述 7 个步骤分别进行说明。

### 步骤 1：获取需求—域建模

域建模是 UML 模型的静态部分的基础。建立域模型时，首先确定真实世界中的抽象，即系统中将涉及的主要概念性对象。设计面向对象的软件时，应根据这些实际问题空间对象设计软件的结构。这里的思想是，与软件需求相比，真实世界发生变化的频率更低。域建模起一个术语表的作用，用例编写者可以在编写用例的早期就使用它。

在确定对象的同时，也可以确定他们之间的关系（泛化和聚集）。域建模的重点是确定对象及他们之间的关系，对于对象的属性和方法，都是以后详细设计的议题。域类的最佳来源很可能是高级问题陈述、低级需求和问题空间的专业知识。要发现域类，首先应从这些来源中挖掘尽可能多的相关陈述（名词和名词短语会成为对象和属性；动词和动词短语会成为操作和关联）。快速地构建域模型，并期望在以后的工作中将其逐步完善。

10 种常见的域建模错误如下。

① 立即给关联指定多重性，确保每个关联都有明确的多重性。不应该在域建模期间就指定多重性，这将占用大量的时间，而且是导致分析瘫痪的主要原因之一。

② 对名词和动词的分析过度，而背离初衷。过度的分析会使设计处于过低的抽象层次，同时有神经崩溃的危险。

③ 不对用例和时序图进行研究，就将操作分配给类。不应该在域建模期间将任何操作分配给类，因为目前还没有足够的信息可以作出正确的决策。

④ 在确保已满足用户需求之前，对代码进行优化以提高重用性。要实现完整性和重用性，需考虑属性和操作，但目前没有这些信息，所以将过多精力用于提供类的可重用性并不明智。

⑤ 对于每个“…部分（part of）”关联，就使用聚集还是组合而争论不休。在域建模期间统一使用聚集即可，至于要区分他们，等到详细设计的时候吧。

⑥ 未对问题空间进行建模前，就假定一种具体的实现策略。在高级类图中，不应该涉及具体的技术内容。

⑦ 将类命名为难以理解的名字，而不是直观名字。

⑧ 直接进入实现结构，如友元关系和参数化类。这些东西与解决方案空间相关，而与问题空间无关，域建模绝对是属于问题空间的。

⑨ 在域类和关系型数据库表之间建立一对一的映射。关系数据库表中的很多属性

不能照搬到对象模型环境中，应该尽可能通过聚集，将属性组转换为“助手（Helper）”类。这种类包含的属性和操作可被组合成更小的“部分（Piece-Part）”类。

⑩ 过早地执行‘模式化’，这将导致根据与用户问题毫无关系的模式创建解决方案。模式通常在健壮性分析的时候才能发现。

## 步骤2：获取需求—用例建模

为新系统创建用例的任务应该是这样完成的：首先确定尽可能多的用例，然后不断地编写和修改描述这些用例的文本。在这一过程中，将发现新的用例和通用情况。确定用例时必须切记一个最重要的原则：他们必须与系统用户手册中的材料密切相关。每个用户和用户指南的各个部分间的关系必须是显而易见的。也就是说，必须从用户的角度出发来设计系统。同时这也是对用例驱动的含义做了总结，即首先编写用户手册，然后编写代码。

用例文本的基本格式应该是‘名词—动词—名词’。当发现新对象的时候，应该及时更新域模型。对于每个用例，尽可能考虑到各种可能的分支——这需要大量的时间。

一些对于用例的建议如下所述。

① 创建一个用例模板，其中只包含“基本流程”和“分支流程”。不要放入其他内容，因为它们只会分散注意力。

② 提出问题“将发生什么？”，进入基本流程。

③ 提出问题“然后将如何？”，不断提出这样的问题，知道获得有关基本流程的所有细节。

④ 提出问题“否则将如何？”，是否还可能发生其他情况？确定吗？不断提出这些问题，直到记录下来大量的分支流程为止。

这里的目标不是构建完美的用例，而是考虑用户可能执行的各种操作。对系统行为的定义越完美，构建系统的工作将越容易。这里，作者还推荐将用例分组分包，因为包为小组间分配工作提供了逻辑边界。同时，一条不错的规则是：每个包应该对应于用户手册的一章或者至少一个大节。

10 种最常见的用例建模错误如下。

① 编写功能性需求，而不是编写使用场景文本。功能性需求通常表述为系统应具备的功能，而使用场景描述的是用户执行操作及系统作出的响应。最终，用例文本将作为系统运行阶段行为的规范（文本将被放置在时序图的左边）。因此，（主动语态的）使用描述（行为）和（被动语态的）系统需求之间必须有清晰的界限。

② 描述属性和方法而不是使用情况。用例文本不应该包含过多的细节，但也不应该没有任何关于屏幕字段的细节。字段名通常直接对应于域类中的属性名，所以，在

用例中不应该命名或描述方法。用例说明的是系统将如何完成工作，而不是系统将完成什么工作。

③ 编写用例过于简洁。对于用例文本来讲，冗长更胜于简洁。用户手册将基于用例文本，对编写用户文档而言，细节过多总比不足强。

④ 让自己与用户界面完全脱离。

⑤ 不给边界对象提供明确的名称。边界对象（Boundary Object）是参与者与之交互的对象，通常包括窗口、屏幕、对话框和菜单。为包含足够的细节并明确指出用户的导航情况，必须在用例文本中明确地为边界对象命名。

⑥ 不从用户的角度进行编写，并使用被动语态。从用户的角度，使用主动语态编写的用例文本的效率是最高的。

⑦ 只描述用户交互，而忽略系统作出的响应。用例描述的是对话双方（用户—系统），以及我们自己试图发现的发生在系统中的所有软件的行为。遗漏系统的响应，也就忽略了软件行为。

⑧ 不描述操作的分支流程文本。

⑨ 不将重点放在用例的内部，而是放在如何到达这里或以后将发生的情况上。Rosenberg 认为冗长的模板是在浪费时间（包含前置条件、后置条件等）。

⑩ 花一个月的时间来决定使用包含结构还是扩展结构。选择哪一种并不重要，重要的是应该选择一种机制，并一直使用它。与一种结构相比，同时使用两种结构更糟糕，因为容易混淆并陷入困境。

### 步骤 3：里程碑 1—需求复核

需求复核旨在确保用例和域模型一起满足客户的功能性需求，同时确保客户知道开发小组将根据这些需求做何种设计。需求复核必须有客户代表、开发小组代表和经理参加，目标是在各方之间就已有的用例、域模型和原型元素达成基本一致，以确定系统的功能需求。

10 种最常见的需求复核错误如下。

① 根本不进行需求复核，而是让编码人员随心所欲地编写。

② 不确保用例文本符合所需的系统行为。

③ 不使用任何 GUI 原型或屏幕模型来帮助验证系统行为。

④ 用例高度抽象，让不懂技术的客户就像在看天书。客户不可能在他不理解的用户例上签字。

⑤ 不确保域模型准确地反映真实世界的概念性对象。

⑥ 不确保用例文本参考域对象。



⑦ 不质疑不包含任何分支流程的用例。90% 以上的好用例都包含至少一个分支流程。

⑧ 不质疑每个用例是否考虑到了所有的分支流程。

⑨ 不管用例是否是用被动语态书写的。

⑩ 不管用例是否占 4 页的篇幅。避免过长的用例模板。

#### 步骤 4：需求的健康检查——健壮性分析

在软件开发中，最难解决的问题之一是，从“什么”视图演变到“如何”视图，健壮性分析正是一种帮助人们完成这项工作的技术。在 ICONIX 过程中，这种简单但很有用的技术在分析（什么）和设计（如何）之间架起了一座至关重要的桥梁。通过健壮性图进行健壮性分析：先绘制健壮性图，仔细检查用例文本，检查每一个句子，然后根据句子的描述绘制出参与者、边界对象、实体对象和控制器及图中不同元素之间的关系；然后复核健壮性图，通过观察用例文本与图是否“图文一致”来发现问题，修改用例文本，当然也要修改静态模型。在这个阶段，应该把一些关键属性添加到类图中。

健壮性图是一个类图，由以下三种图例组成。

① 边界对象（Boundary Object）：参与者使用它来与系统交互，通常包含窗口、屏幕、对话框和菜单。可以从 GUI 原型和用例文本中找出边界对象。

② 实体对象（Entity Object）：通常是来自域模型中的对象，常对应于数据表 and 文件。

③ 控制对象（Control Object）：将边界对象和实体对象关联起来（通常叫控制器，因为它们一般不是真正的对象）。它包含大部分应用程序逻辑，如经常修改的业务规则和策略。控制器偶尔会是设计中的“真正的对象”，然而大部分时间里，它只是一个占位符，用于避免遗漏用例要求的任何功能和系统行为。

在初步设计阶段，应仔细考虑各种可供选择的设计策略和技术体系结构。此时，应开始找出与系统性能相关的问题。在健壮性分析期间，将以需求级用例文本为基础，做出一些初步设计方面的假设。

对健壮性图，必须遵守如下 4 条规则。

① 参与者只与边界对象交互。

② 边界对象只能与控制器和参与者交互。

③ 实体对象只能与控制器交互。

④ 控制器可与边界对象、实体对象及其他控制器交互，但不能与参与者交互。

健壮性分析要素如下。

(1) 它是对用例文本的一次健康检查，帮助确保用例文本的正确性，且没有指定



不合理或不可能的系统行为。这种改进使用例文本的特性从纯粹的用户手册角度变为对象模型上下文中的使用描述。

(2) 帮助确保用例考虑到了所有必须的分支流程。把这件事情放在健壮性图上完成,将使时序图的绘制可以节省 3~4 倍的时间。

(3) 发现在域建模中遗漏的对象,发现对象命名冲突等问题。同时确保在绘制时序图之前确定了大部分的实体类的边界类。

(4) 缩小分析和设计之间的鸿沟,从而完成系统的初步设计。

10 种最常见的健壮性分析错误如下。

① 违反一种或多种健壮性分析规则。

② 不使用健壮性分析来帮助我们在用例文本中采用一致的格式。通过健壮性分析,可以确保用例文本都是以‘主—谓—宾’的句子格式描述的。

③ 健壮性图中不包含分支流程。

④ 不使用健壮性分析来确保类图和用例文本中的类名一致。

⑤ 给健壮性图中的类分配属性(目前还没有足够的信息,应该等到时序图的时候)。

⑥ 包含的控制器过多或过少。Rosenberg 建议每个图包含 2~5 个控制器,如果控制器超过 10 个,则应该考虑将用例分解。

⑦ 试图使健壮性图十全十美,而花费过多的时间。健壮性图只充当辅助引擎,它驱动用例向前发展,成为一个 OO 的设计方案。当完成这个任务后,就不需要它了。

⑧ 试图在健壮性图上完成详细设计。临时图的概念对初步设计很有用,但对详细设计没有任何用处。

⑨ 不在用例文本和健壮性图之间进行可视化跟踪。

⑩ 不更新静态模型。

### 步骤 5: 里程碑 2—初步设计复核

初步设计复核(PDR)指的是对要构建的每个场景的健壮性图和用例文本进行复核,确保它们一致,并完善、正确地表现了系统的行为。PDR 也应该有客户代表、开发小组代表和经理参加。一个重要的区别是,这是客户修改其需求的最后机会,之后开发人员根据指定的用例向前推进,最终编写出代码。PDR 是一个分水岭,过了该分水岭之后,客户的主动参与将不再受欢迎。

10 种最常见的 PDR 错误如下。

① 不确保客户知道这是他们修改行为的最后机会,之后将开始构建系统。

② 不确保用例文本和健壮性图之间的一致性。

③ 不确保将新的实体对象加入到域模型中。

④ 不考虑域类的属性。对一组用例进行健壮性分析时，应获得域模型中的类的完整属性集。

⑤ 期望 PDR 期间已经将操作分配给了类。

⑥ 不再次向客户重申，用例文本是开发人员和客户之间的契约。

⑦ 要求在初步静态设计中大量使用设计模式。找出健壮性图中的模式，尤其是那些容易对应到已有的设计模式的模式或自己发明的模式是对的。但将健壮性图中出现的简单的初步设计模式扩展为详细设计模式却是错误的。后一种工作应该留到时序图和高级类图去完成。

⑧ 不对健壮性分析的名词、动词规则进行复核。

⑨ 期望健壮性图中包含完整的详细设计而不是概念设计。

⑩ 仔细复核健壮性图中每个箭头的方向，而不是进行快速跟踪以核实是否考虑到了所有的行为。

#### 步骤 6：详细设计—时序图

健壮性分析旨在发现对象，而详细设计则主要是关于分配行为：将确定的软件操作（函数、方法）分配给发现的一组对象。时序图是详细设计的核心，也是系统动态模型的核心。

ICONIX 方式绘制时序图的步骤如下。

(1) 复制用例文本，粘贴到页面的左边。

(2) 加入健壮性图中的实体对象。

(3) 加入健壮性图中的边界对象和参与者。

(4) 将方法分配给类。这时需要将健壮性图中的控制器转换为一组执行所需行为的方法和消息（有时候，控制器也可能转换为一个控制对象）。

10 种最常见的时序图的错误如下。

① 不为每个用例绘制一个时序图。只有为每个用例中的所有事件流程绘制交互图后，才能确定自己找出系统要求每一个对象扮演的角色，即每个对象的职责。

② 不将用例文本添加到时序图中。

③ 不首先在健壮性图上确定所有必须的对象。

④ 不在用例文本和消息箭头之间提供可视化跟踪。

⑤ 不说明对象间的通信管道，让时序图表示高级抽象。健壮性图上无须说明对象间的通信管道，因为它们反映的是初步设计。然而编码将根据时序图来完成，因此需要极其详细地说明实际的设计方案。

⑥ 将时序图绘制成流程图，而不是使用它在对象间分配行为。

⑦ 不将重点放在有趣的方法上，而是放在 `getter`、`setter` 上。

⑧ 不仔细考虑消息箭头的起点（即特定时间内，哪个对象拥有控制权）。

⑨ 通过绘制消息箭头分配行为时，不遵循职责驱动的 OOD 这一基本原则。对象只能有一种“性格”，应避免对象有“精神分裂症”。常常自问：该方法分配给该对象是否是最合适的？该方法完成的任务是否与该对象明显有关？

⑩ 不为每个用例包绘制本地类图，以更新静态模型。绘制本地化静态类图，以列出解决方案空间对象和问题空间对象是一个不错的主意。一条不错的指导原则是，对每一个用例包，绘制一个本地类图。静态模型可能很庞大，无法在一张易于阅读的图中完全表现出来。

### 步骤 7：里程碑 3—关键设计复核

关键设计复核（CDR）旨在确保详细设计的“如何”与用例指定的“什么”完全一致，同时确保详细设计足够详细，并可根据它来编写代码。对于 CDR，差不多所有的设计人员和开发人员都必须参加。

10 种最常见的 CDR 错误如下。

① 邀请不懂技术的客户参与这种设计复核。

② 不对照时序图仔细地检查用例文本。

③ 不仔细检查每个时序图中所有消息箭头的起点和终点。对于不连贯的时序图，没有办法用于指导编码。

④ 复核时序图时，不仔细考虑面向对象的设计原则。

⑤ 不根据“高品质类”标准对静态模型进行复核。

⑥ 不考虑细节。

⑦ 不考虑设计模式在设计中是否有用。

⑧ 允许漠视诸如 DCOM 或 EJB 等实现技术的“总体”时序图的存在。此刻应该尽可能缩短从详细设计进入实现时需要跨越的距离。

⑨ 不复核当前版本中要构建的所有场景的时序图。

⑩ 编写代码之前不关心细节，认为重构代码可以修复所有的问题。

ICONIX 提倡，在项目开始阶段，就应该构建域模型和用例模型，其中用例模型驱动了整个动态模型，而域模型则驱动着整个静态模型。从另一个角度看，动态模型又驱动静态模型的完善。所以，整个系统都是由用例直接或间接驱动的。用例驱动，即软件的体系结构和设计方案都是通过分析使用场景推断出来的。

### 6.6.2 用例建模

用例的概念既包括用例图 (Use Case Diagrams), 还包括用例文档。

用例图以旁观者的视角描述对系统的印象。强调这个系统“做什么”而不是这个系统“如何做”。

用例图与场景密切相关。场景 (Scenario) 是指当用户与系统进行交互时的情形。下面是一个医院门诊部的场景:

“一个病人打电话给诊所, 预约一年一次的体检。接待员在预约记录本上找出最近的没有预约过的时间, 并在那个时间登记预约记录。”

用例 (Use Case) 是完成一个任务或者实现一个目标的若干场景的概要描述。角色 (Actor) 是发起该任务的相关事件的人、部门或系统。图 6.44 所示是一个体检诊所“预约门诊”的 (Make Appointment) 用例图。角色是病人。角色与用例的关联是通信关联 (Communication Association, 简称通信)。

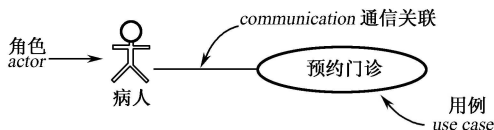


图 6.44 用例图的基本构成

角色是人形的图标, 用例是一个椭圆, 通信是连接角色和用例的直线。

用例图是角色、用例及其之间的关联的集合。在图 6.45 中, 可以看到, “预约”用例是含有四个角色和四个用例的一个完整用例图的组成部分。注意, 一个单独的用例可以与多个角色关联。

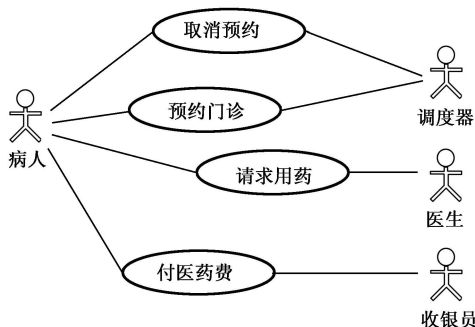


图 6.45 诊所体检的用例图

用例图有如下三个重要作用。

- (1) 确定需求。当系统已经分析完并且设计成型时，用新的用例产生新的需求。
- (2) 与用户沟通。开发者和用户都能够看懂用例图，便于互相交换意见。
- (3) 产生测试用例。根据一个用例的多个场景，可以提出一套与这些场景相关的测试用例。

随着分析的深入，可以对用例图进行扩展，添加特征，增加需求，以得到更加详细的用例图。图 6.46 所示就是对图 6.45 的扩展。

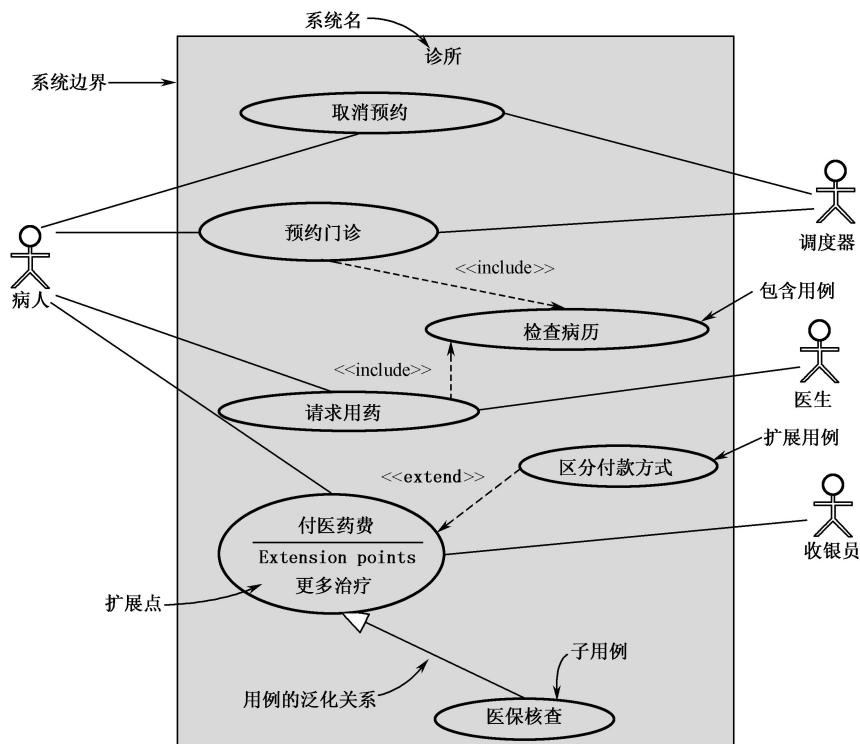


图 6.46 扩展后的用例图

新增加的矩形框叫做系统边界，系统边界内为诊所系统，系统边界外是不同的角色。系统边界将系统与角色分离开来。

用例的泛化表现的是用例之间的继承关系。“付医药费”是父用例，而“医保核查”是子用例。如果必要，则子用例随时都可以被其父用例替换。泛化的符号是空心三角箭头，由子用例指向父用例。

包含关系 (Include) 体现了用例之间的分解关系。当一个用例中包含有另一个用例的一些步骤时, 则这个用例就应该包含另一个用例。包括关系尤其有助于将一个用例分解为两个或两个以上不同的用例。例如, “预约门诊” 和 “请求用药” 与 “检查病历” 之间属于包括关系, 前两者都把 “检查病历” 作为子任务。包括关系的符号为虚线箭头, 从基本用例指向所包括的用例, 虚线上标记为 “<<Include>>”。

扩展关系 (Extend) 表示的是一个用例是另一个用例的变型。当第一个用例给第二个用例增加了步骤, 则称第一个用例扩展了第二个用例。扩展关系的符号为虚线箭头, 箭头上标记了 “<<Extend>>”, 从扩展用例指向基本用例, 基本用例中将写上扩展点 (Extension Point), 表示什么时候扩展用例比较合适。

### 6.6.3 结构建模

结构建模就是对对象中的静态关系进行建模, 包括类图、包和对象图、组件图和配置图等。

#### 1. 类图 (Class Diagram)

类图通过显示系统的类及这些类之间的关系来表示系统。类图是静态的, 它只表示出类之间的相互影响是什么, 但不会显示出类在相互影响时会发生什么。

图 6.47 描述的是一个顾客从零售商品目录上订购商品的类图。主要的类是 “订单” (Order)。与之关联的类是 “顾客” (Customer) 和 “付款” (Payment)。“付款” 有三种形式: “现金” (Cash), “支票” (Check), “信用卡” (Credit)。“订单” 包括 “订单明细” (Order Details), 角色名为 “列项” (Line Item), 每个 “订单明细” 都与 “订单项” (Item) 关联。

在 UML 中, 类的符号是一个矩形框, 分为三个部分: 类名、属性和操作。抽象类的名字是斜体的, 如 Payment。类之间的关系是连接线。

类图有三种关系。

① 关联 (Association): 表示两个类的实例间的关系。如果一个类的实例必须要用到另一个类的实例才能完成工作, 则这两个类的实例之间就存在关联。在类图中, 关联的符号是两个类之间的连线。

② 聚合 (Aggregation): 表示两个类之间是部分与整体的关系。一个类属于另一个类时, 两个类之间就存在聚合关系。聚合的符号是一个带菱形的连线, 菱形一端指向具有整体性的类。聚合关系与图 6.19 中的组装关系语义一致, 但是图形符号不同。在图 6.47 中, “订单” (Order) 类是 “订单明细” (Order Details) 类的容器。

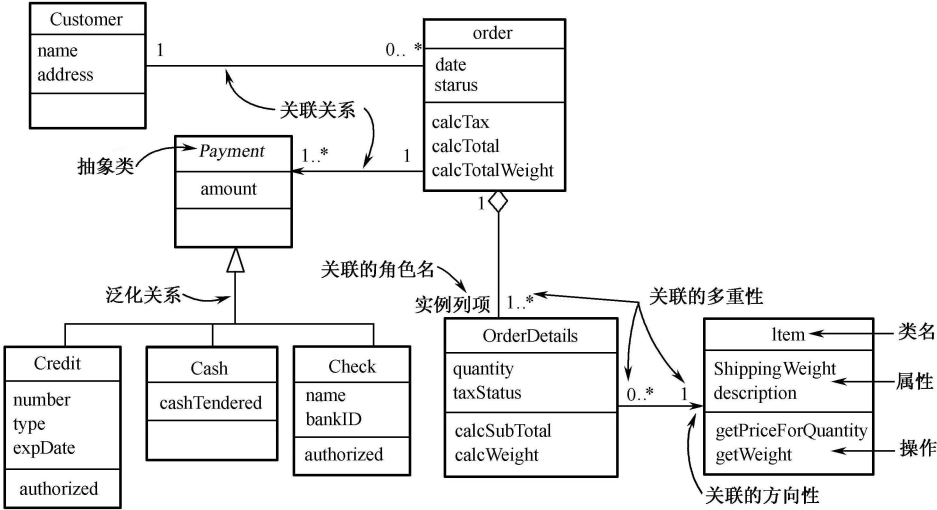


图 6.47 类图中的各种要素和关系

③ 泛化（Generalization）：表示一个类是其他类的超类。泛化关系的符号是一个指向超类的空心三角形。例如，“付款”（Payment）是“现金”（Cash）、“支票”（Check）和“信用卡”（Credit）的超类。

每个关联有两个端点。每个端点用一个角色名（Role Name）来说明关联的作用。例如，一个“订单明细”（OrderDetails）实例是一个“订单”（Order）实例的列项（Line Item）。

关联上的方向性（Navigability）箭头表示该关联传递或查询的方向。“订单明细”（OrderDetails）类可以查询相应的“订单项”（Item），但不可以反过来查询。箭头方向同样可以说明类“拥有”这个关联的实现；既然如此，“订单明细”（OrderDetails）即拥有“订单项”（Item）。没有方向性箭头的关联则是双向的。

多重性（Multiplicity）用关联端点上的数字表示，表示一个类的实例可能关联的其他类的实例的个数。多重性可以是一个数值，也可以是一个取值范围。多重性的图示在前面的图 6.20 中已有描述。在图 6.47 中，每个“订单”（Order）只有一个“顾客”（Customer），但一个“顾客”（Customer）可以有任意多个“订单”（Order）。表 6.1 列出了最常见的多重性示例。

表 6.1 最常见的多重性示例

多重性	语 义
0..1	0 或 1 个实例。n..m 符号表示有 n~m 个实例
0..* 或 *	没有实例个数的限制（包括没有）
1	只有一个实例
1..*	最少一个实例

每个类图包括类、关联和多重性。为了使图示更清楚，方向性和角色是可选项。

(1) 关联的说明：聚集和组合

表示整体和部分关系的关联是聚集。组合是强关联，表示部分属于仅有的一个整体，如果整体不存在，则部分也就不存在。组合的符号是一个实心菱形，从整体端指向部分端。如图 6.48 所示，“包厢”属于某个“电影院”，是组合关系，如果电影院被毁了，则包厢也就不存在了。反之，“电影”和“电影院”的关系就没有限制得如此紧密，它们是聚集关系。

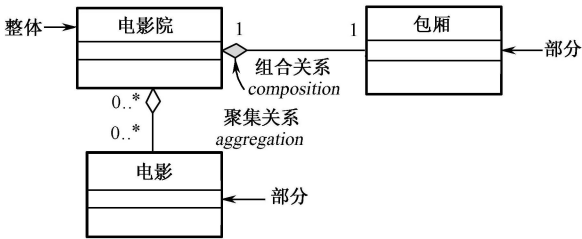


图 6.48 聚集和组合

(2) 类的说明：可见性与范围

类的符号由三个部分组成：类名、属性和操作。类的属性和操作将按照访问和范围来标记。图 6.49 所示为扩展的“订单”类图。

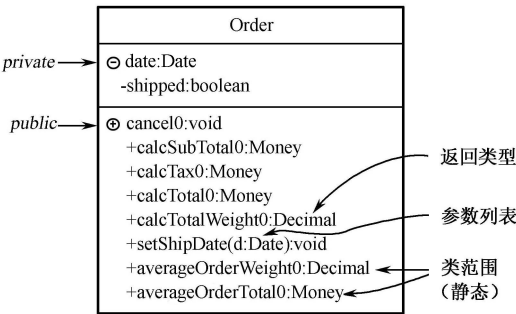


图 6.49 类的各部分



图中的标记按照 UML 的约定。

- ① 静态成员使用下划线，实例成员不使用下划线。
- ② 操作的形式：〈访问权限修饰词〉〈类名〉（〈参数列表〉）：〈返回类型〉。
- ③ 参数列表用冒号置于每个参数类型之前来表示。
- ④ 访问权限修饰词置于每一个成员之前。

符号	访问权限
+	public
-	private
#	protected

（3）类的说明：依赖与约束

依赖（Dependency）表示两个类之间的关系，其中一个类的改变会使得另一个类也改变。依赖的符号是虚线箭头，指向被依赖的类。例如，在图 6.50 中“实习”（Co\_op）依赖于“公司”（Company），如果要改变“公司”，则必须也改变“实习”。

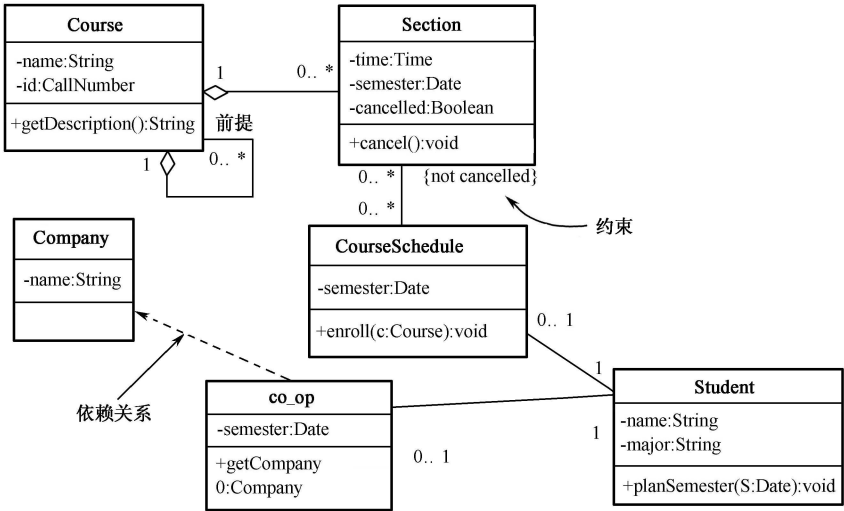


图 6.50 依赖与约束

约束是每一个设计出来的实现都必须满足的条件。约束写在“{ }”内，在图 6.50中，约束表示“章节”（Section）可以是“课程表”（CourseSchedule）的一部分，条件是它没有被取消。

（4）类的说明：接口与版型

接口是一组操作签名。在 C++ 中，接口由仅有一个纯虚拟成员的抽象类实现。在 Java 中，则直接实现。

图 6.51 所示的类图是一个专业会议的模型。描述会议重要性的类有“会议议程” (SessionTalk) 和“会议” (Session), “会议”有一天的“会议议程”。“往返时间表” (ShuttleSchedule) 及其“往返地点” (ShuttleStop) 列表对住在远处旅馆的与会者非常重要。该图有一个约束, 即往返地点是已经安排好的。

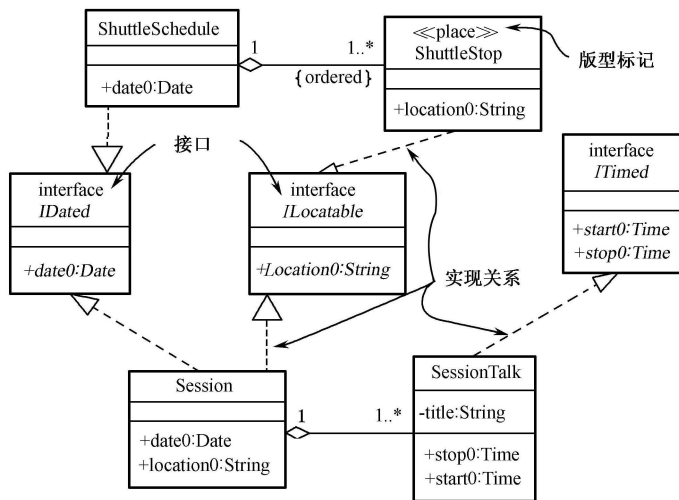


图 6.51 接口与版型

图中有三个接口: IDated, ILocatable 和 ITimed。接口名均以字母“I”开头。接口名及其(抽象)操作以斜体书写。

例如,“往返地点”(ShuttleStop)类,其操作与接口 ILocatable 匹配,表示接口的执行(Implementation)或实现(Realization)。

“往返地点”类具有版型<<Place>>。所谓版型,就是 UML 提供的一种扩展方法,用来从已有类型创建新的模型元素。版型名通常写在类名上方,用尖角双括号“<< >>”表示。接口是一种特殊的版型。

UML 中有两种标记接口的符号,一种符号如图 6.51 所示,另一种可使用棒棒糖符号或圆圈符号,如图 6.52 所示。

用圆圈符号表示接口时,接口是圆圈,连线的另一端是实现的类。由于这种表示比较简洁,略去了较多细节,因此简化了原图。

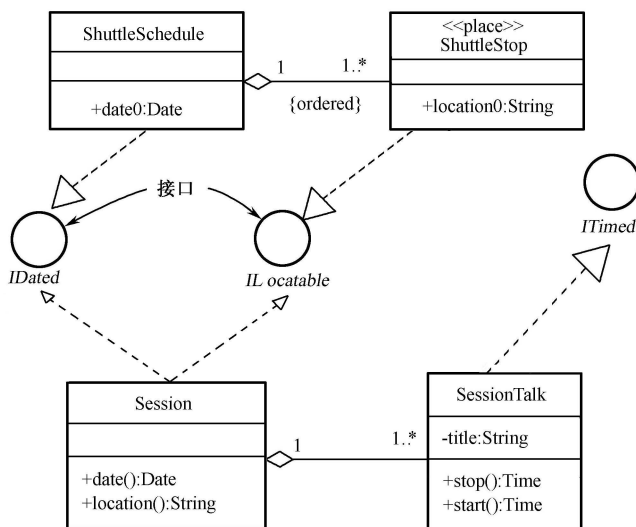


图 6.52 接口的另一种表示

## 2. 包和对象图

为了简化复杂的类图，可以把类用包（Package）来组织。包是逻辑上相关的 UML 元素的集合。图 6.53 所示是一个业务模型，其中类被用包分组。

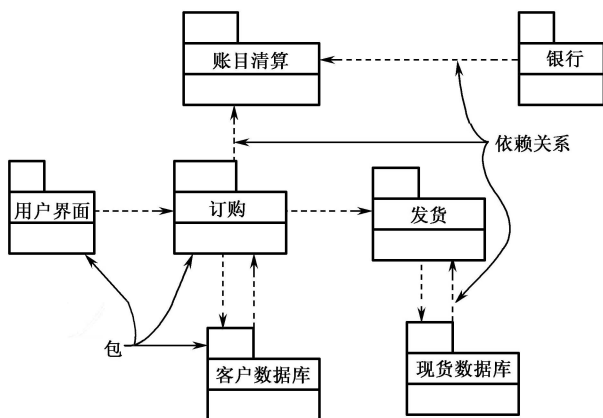


图 6.53 包图

包用矩形表示，矩形的顶部带有小标签。包名写在标签上或者矩形里面。虚线箭

头表示依赖关系。如果第一个的包改变可能会导致第二个包的改变，则第二个包依赖于第一个包。

对象图（Object Diagrams）用来表示类的实例。对象图有助于解释具有复杂关系的细节（特别是递归关系），如图 6.54 所示。

图 6.54 表示某大学的“部”（Department）可以包括很多的其他“系”（Department）。

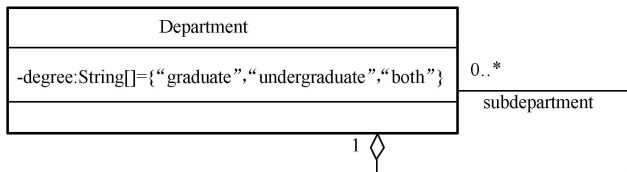


图 6.54 用类图表示递归

图 6.55 所示，是图 6.54 所示类图的实例，其中有了若干具体的例子。

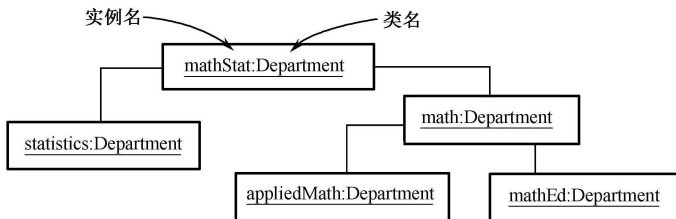


图 6.55 对象图描述类的复杂关系

每个类图的矩形对应了一个单独的实例。在 UML 图中，实例名带有下画线。只要意思清楚，类或实例名可以在对象图中被省略。

### 3. 组件图和配置图

组件图描述了类图的物理实现，其中的组件（Component）是代码模块。

配置图（Deployment Diagrams）则显示软件及硬件的配置，反映了系统的物理结构。

图 6.56 描述了某房地产交易系统的软、硬件组成关系。

物理硬件使用节点（Node）表示，每个组件属于一个节点。组件用左上角带有两个小矩形的矩形表示。

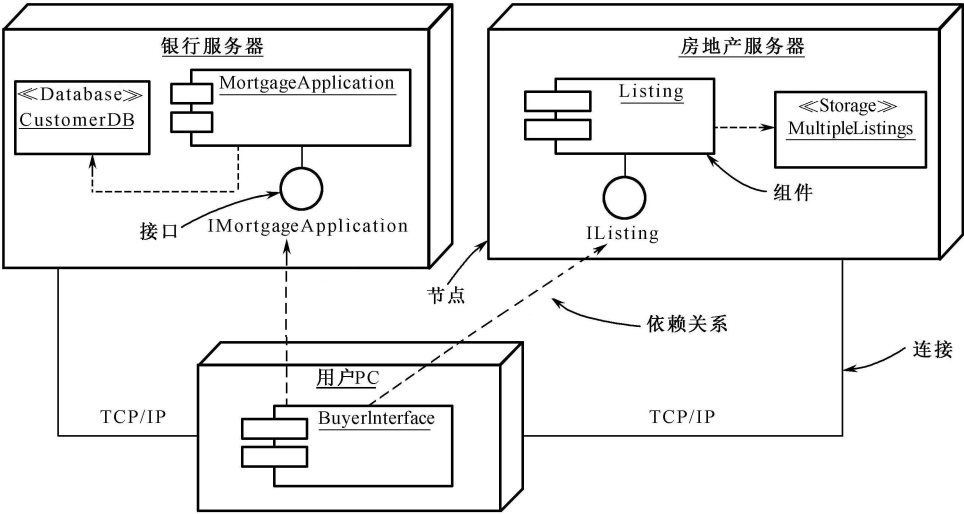


图 6.56 组件图和配置图

6.6.4 行为建模

顺序图、协作图、状态图和活动图为模型建立了动态视图。这些图使得开发人员能够洞察模型内部的行为机制。其中，顺序图和协作图侧重于完成单个过程的消息的描述，状态图则侧重于对单个对象的描述。活动图则侧重于执行单个任务时的活动流的描述。

1. 顺序图

类图 and 对象图均属于静态模型视图。交互图则是动态的，因为交互图描述了对象如何协同。

顺序图是一种交互图，它详细描述了操作如何执行——发送了什么消息？何时发送的消息？顺序图按照时间顺序来组织。操作中所涉及的对象按照其参与消息序列的时序，从左至右列出。

图 6.57 所示是一个预定旅馆房间的顺序图。启动消息序列的对象是“预定窗口”（a Reservation Window），“预定窗口”对象发送“makeReservation（）”消息给“旅馆连锁联盟”（HotelChain）。“旅馆连锁联盟”（HotelChain）接着发送“makeReservation（）”消息给“旅馆”（Hotel）。

如果“旅馆”（Hotel）有空房间，就“下订”（Reservation）并“确认”（Confirmation）。每一条垂直虚线是一条生命线（Life Line），表示对象存在的时间。箭头表示消息

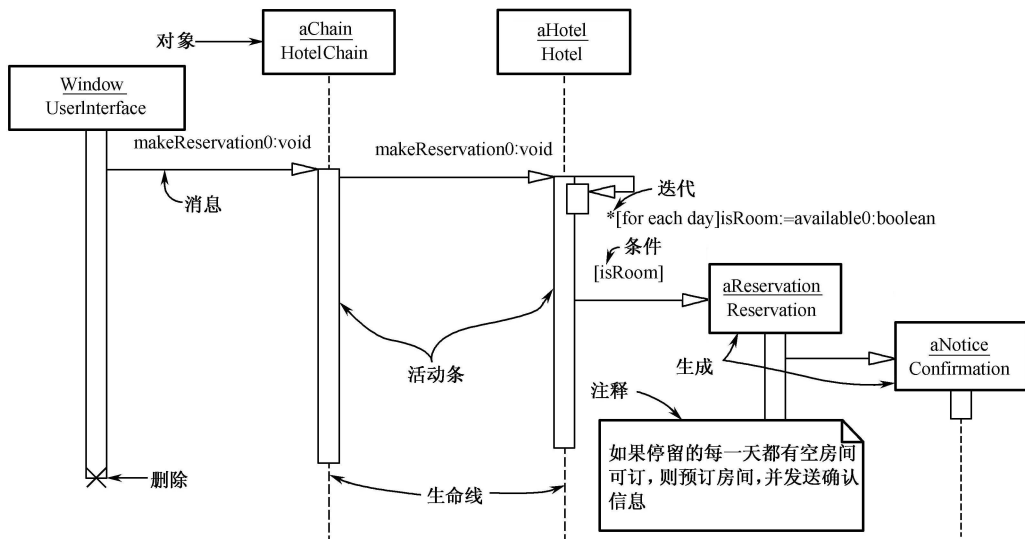


图 6.57 预订旅馆房间的顺序图

调用。消息箭头从发送者出发到接收者生命线上消息的“活动条”（Activation Bar）的顶端。活动条表示消息执行的持续时间。

在图 6.57 中，“旅馆”（Hotel）对象发给自身消息，以确定是否有空房间。如果有空房间，则“旅馆”（Hotel）对象就创建“预定”（Reservation）和“确认”（Confirmation）两个对象。自身消息上的 \* 表示迭代（落实每天旅馆有空房间）。表达式使用方括号“[]”，表示条件。图中还有注释符号，用内折一角的矩形表示。注释可以在任何 UML 图中使用。

顺序图中可以对异步或并发活动进行建模。

异步消息是指当发送者所发送的消息还在处理的时候，该发送者继续发送的额外消息。异步消息的时限与干预消息（Intervening Message）的时限无关。

图 6.58 所示的顺序图描绘了某体检诊室的护士请求诊断检测的行为。“护士”（Nurse）发送了两条异步消息：

- ① 询问“体检诊室”（MedicalLab）以预约测试的日期；
- ② 询问“保险公司”（InsuranceCompany），请求批准检测。

这两条消息的发送和完成的顺序是互不相关的。如果“保险公司”（InsuranceCompany）同意检测，则“护士”（Nurse）将根据“体检诊室”（MedicalLab）提供的日期来安排检测。

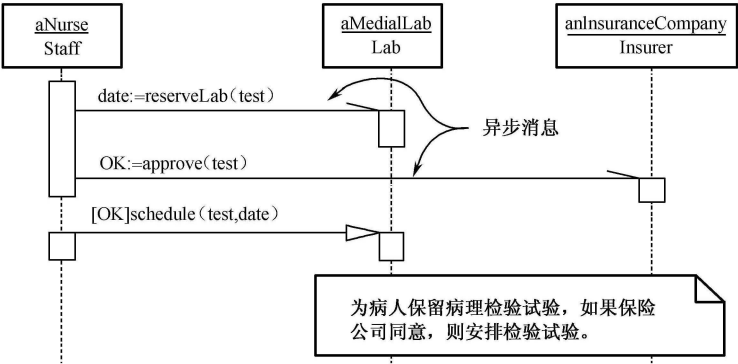


图 6.58 在顺序图中建模异步消息

UML 提供了如表 6.2 所示的消息的规范。

表 6.2 UML 的消息规范

符 号	语 义
————>	简单消息，可能同步或异步
<-----	简单消息返回（可选）
————>	同步消息
——or——>	异步消息

2. 协作图

协作图也是交互图的一种。与顺序图类似，协作图也传递相同的信息，但协作图不描述什么时候消息被传递，只描述对象的角色。在协作图中，对象的角色排列与顺序图不同，不必排列在图的顶部，对象角色矩形上标有类或对象名（或者都有），类名前面使用冒号“:”。协作图的消息用加标签的箭头表示，每个消息都有消息的顺序编号。顶层消息的数字是 1。同一顺序等级的消息（也就是同一个调用中的消息）有同样的数字前缀，再根据它们出现的顺序增加一个后缀 1、2 等，如图 6.59 所示。

3. 状态图

对象拥有行为和状态。对象的状态是由对象当前的活动和条件决定的。状态图显示出了对象可能的状态及由状态改变而导致的状态转移。

图 6.60 所示描述了一个在线银行系统的登录部分。登录过程包括输入合法的社会保障号码（SSN）和身份证号码（PIN），再提交给系统验证信息。

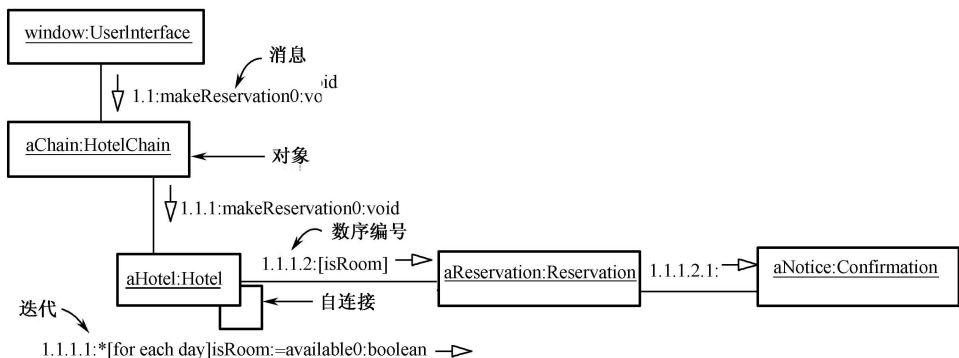


图 6.59 协作图的表示

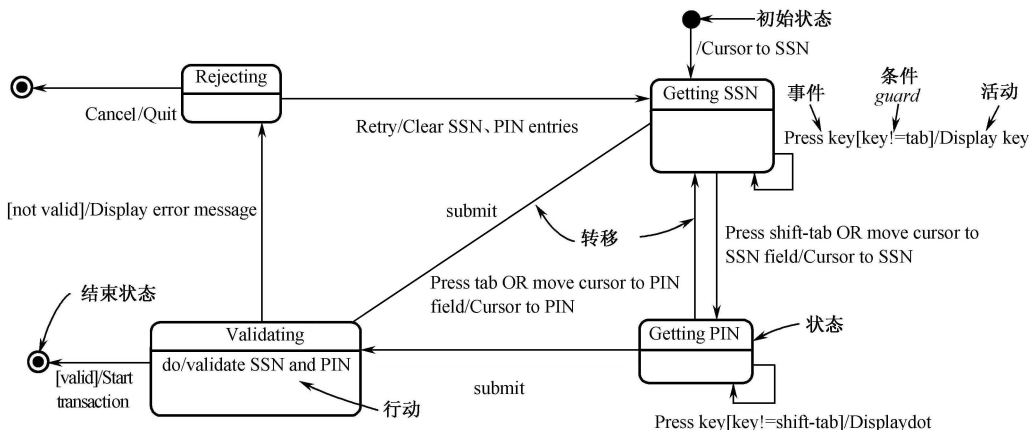


图 6.60 状态图

登录部分可以被分解为四种不重叠的状态：“获取社会保障号码”（Getting SSN）、“获取身份证号码”（Getting PIN）、“验证”（Validating）及“拒绝”（Rejecting）。每个状态都有一套完整的转移（transition）来决定后续状态。

状态是用圆角矩形来表示的。转移用箭头表示，从一个状态指向另一个状态。触发转移的事件或条件写在箭头的旁边。图 6.60 中有两个自身转移，一个在“获取社会保障号码”（Getting SSN），另一个在“获取身份证号码”（Getting PIN）。

初始状态（黑色圆圈）是虚拟初始状态。结束状态也是虚拟动作结束。

事件或条件触发动作时用“/行动”表示。当进入“验证”（Validating）状态时，对象不需要等外部事件来触发转移，而是执行一个活动。活动的结果将决定后续状态。

状态图中也可以对并发和异步进行表示。



状态图中的状态可以嵌套。相关的状态可以组合在一起而形成一个组合状态。当活动涉及并发的或异步的子活动时，状态嵌套就很有必要。图 6.61 描述了一个拍卖过程的状态图。

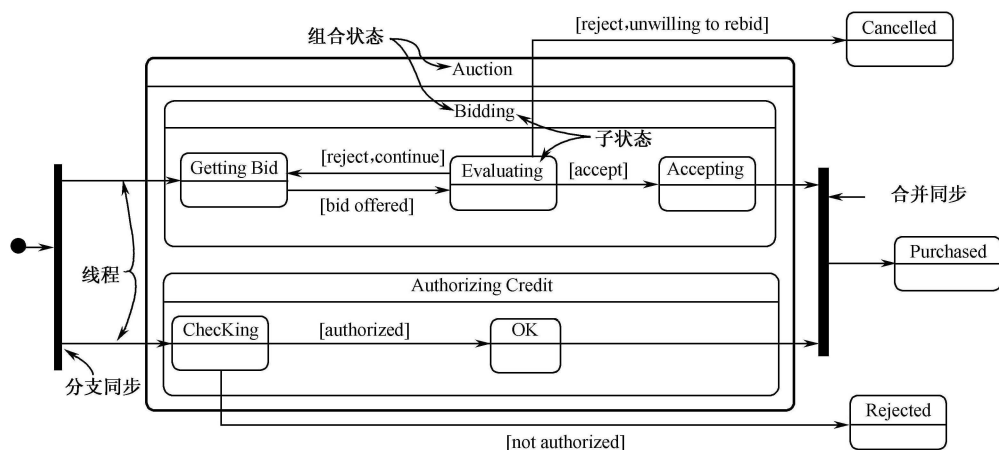


图 6.61 状态图中对并发和异步的表示

这个拍卖过程命名为一个组合状态“拍卖”（Auction），其中包含两条并发的路线，即两个子状态：“出价”（Bidding）和“授信”（Authorizing Credit）。而“出价”（Bidding）状态本身又是一个包含三个子状态的组合状态，“授信”（Authorizing Credit）状态则包含两个子状态。

一进入“拍卖”（Auction）状态，就通过分支同步（Fork），分别进入两条并发的路径。除非出现异常退出（取消或拒绝），否则要退出“拍卖”（Auction）这个组合状态，必须是其中的两个子状态都已经退出。

#### 4. 活动图

活动图从本质上讲是变型的流程图。活动图和状态图之间是有关系的。状态图关注经历一个过程的一个对象（或把过程作为一个对象考虑），而活动图则关注一个过程中的活动流。活动图描述的是活动之间的依赖关系。

设某过程如下：“通过 ATM 从银行账户中取钱。”

该过程中涉及三个类：“顾客”（Customer）、“自动取款机”（ATM）和“银行”（Bank）。活动图中可以划分出许多对象泳道（Swimlane），每个泳道的顶部写上对象所属的类名，表示哪些对象负责哪些活动。活动用圆角矩形表示。每个活动都有一个单独的转移（Transition）连接到其他的活动。整个过程从顶部的黑色圆圈开始到底部的

黑白的同心圆圈结束，如图 6.62 所示。

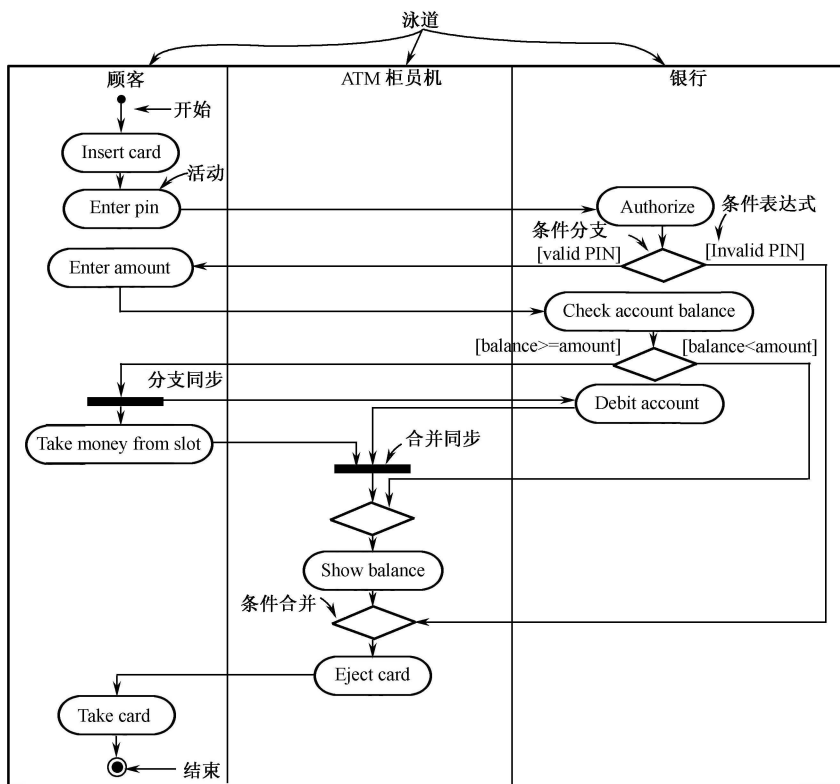


图 6.62 活动图示例

转移可以分支 (Branch)，成为两个以上的互斥的转移。条件表达式 (Guard Expression) (在 [ ] 中) 表示转移是从一个分支中引出的。分支及分支结束时的合并 (Merge) 在图中用空心菱形表示。

转移也可以分支同步 (Fork)，成为两个以上的并行活动。分支同步及分支结束时的合并同步 (Join) 在图中用实心黑线表示，称为同步条。

## 6.7 UML 建模工具简介

### 6.7.1 概况

随着 UML 的成功应用，越来越多的组织、机构都开展了对 UML 建模工具的研制活动。目前，UML 建模工具的数量已经超过上百种。

按照这些工具的特点，大体可将其分为两类：商业工具与开源工具。有些商家为了推广自己的产品，在推出商业工具的同时，也发布限制部分功能的免费版，其主要目的是提供给在校学生和研究人员使用。因此对用户而言，可供选择的 UML 建模工具的范围就大大扩展了。

目前，通过分析众多 UML 建模工具的主要功能，可以归结出以下共同特点：

- ① 支持 UML 规范的新版本；
- ② 支持可视化的建模环境；
- ③ 支持正向工程与反向工程，支持多种编程语言，并与编程环境集成；
- ④ 支持基于 XML 的模型文件交换；等等。

一般地，可以使用表 6.3 中的比较项对 UML 工具进行比较。

表 6.3 UML 建模工具功能比较表项①

版本	支持的图形								代码生成时支持的编程语言				运行平台的支持
	用例图	顺序图	协作图	类图	活动图	状态图	组件图	部署图	必须		其他		
1.0 1.1 1.3 2.0 2.1									CO RB A— ID L	C+ +	JA VA	ActiveX/COM 组件 DDL-SQL MS-COM 接口 PowerBuilder SmallTalk Visual Basic Ada95 Raven C# gcc/pgcc (Linux) J# Ada83 und 95 ANSI-C SPARK83 & 95 Delphi Forte Tool Object COBOL SmallTalk	
													AIX DEC VMS HP-UX Linux OS/2 Solaris Win 95/98/NT 4.0/2000 Win XP/Vista Windows 3.1x Apple Macintosh 68020, System 7.1+ UNIX/Mot if SGI IRIX

6.7.2 几种典型的 UML 建模工具

在商业工具中，最著名的是 IBM 公司的 RSA（Rational Software Architect）。RSA 可以说是 UML 开发工具中最著名的品牌。RSA 的前身是 Rational 公司推出的

① Mario Jeckle. Diese Liste enth? lt inzwischen Aussagen zum Leistungsumfang von mehr als 100 UML-Werkzeugen!. <http://www.jeckle.de/umltools.htm>.

Rational Rose。目前 RSA 是 IBM 公司的产品了。RSA 与 Eclipse 开发环境配合得非常完美，是 IBM 推动 MDA 的两个主力产品。在 2005 年，RSA 也与 DoDAF 相结合，强化了 Real-time 系统建模及大型系统整合设计的环境。

其次是 I-Logix 公司的 Rhapsody。Rhapsody 是 I-Logix 公司推出的 UML 2.0 开发环境。自从 1987 年，I-Logix 就专注于 Real-time 及嵌入式系统的建模 (Modeling)，所以 Rhapsody 是当今嵌入式领域中功能最完善的 UML 开发环境。更值得一提的是，Rhapsody 积极将 DoDAF (Department of Defense Architecture Framework) 应用于嵌入式系统的软、硬件整合设计上。

再次是 Spar 公司的 EA (Enterprise Architecture)。EA 是当今市面上最平易近人的 UML 2.0 开发环境，是澳大利亚的 Sparx 公司推出的产品，它具有以下特色：价钱非常便宜，每份 (License) 大约 2000 元人民币；工具很小，启动速度快，系统很稳定，画面清爽；开放的接口，用 Visual Basic .Net 就能编写出各种各样的外挂程序；充分支持 UML 2.0 和 MDA (Model-Driven Architecture)。

在开源工具中，目前最受欢迎的是一款韩国软件 StarUML。StarUML 可视化建模功能全面，采用便于扩展功能的组件 (plugin) 架构，支持 UML 2.0 和 MDA，提供多种编程语言的代码生成及逆向工程功能。StarUML 为初学 UML 设计的用户提供了一项免费、好用又完整的学习环境。图 6.63 所示为 StarUML 的界面截图。

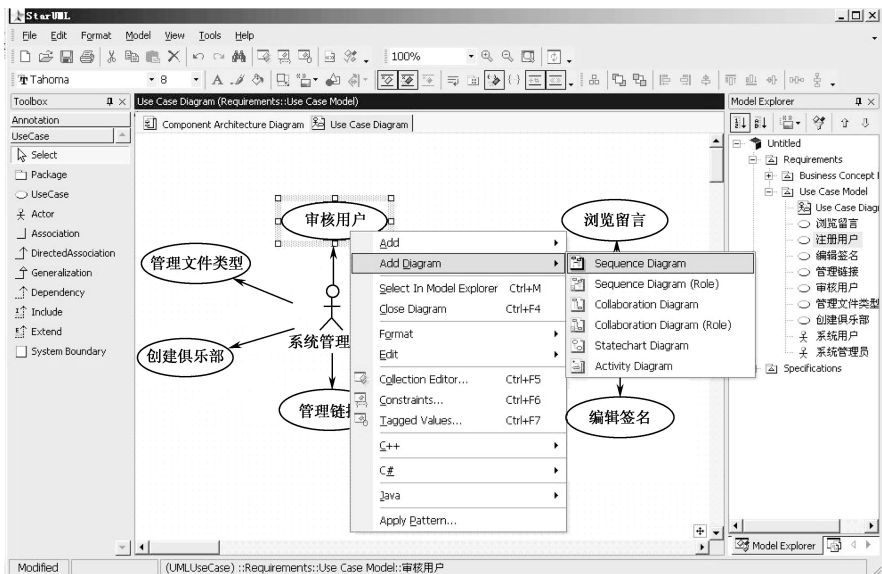


图 6.63 用例图，以及从用例图生成顺序图等界面

## 习 题

1. OOA 方法的五个步骤是什么？
2. OOD 通常包括哪几个部分的设计？
3. 怎样评价面向对象设计方法的质量？
4. 请谈谈对结构化分析设计与面向对象分析设计的认识和比较。
5. 根据下列需求画出液晶彩电的用例图（某些需求属于非功能性的，不一定能够变为用例，且某些需求可能产生不止一个用例）。

需求 1：用户通过按下系统上的按钮或使用遥控器就能够启动系统。

需求 2：用户通过按下系统本身的按钮或使用遥控器就能够更换频道。

需求 3：系统将通过按系统自己的按钮，或者通过遥控器对任何电视频道进行调整。

需求 4：系统有一个内置调谐器并连接一个外部调谐器。

需求 5：系统能显示高清 HD 1080 逐行扫描节目内容，分辨率达到宽屏模式下  $1920 \times 1080$ 。

需求 6：系统能显示高清 HD 1080 隔行扫描节目内容，分辨率达到宽屏模式  $1920 \times 1080$ 。

需求 7：系统能显示高清 HD 720 逐行扫描节目内容，分辨率达到宽屏模式下  $1280 \times 720$ 。

需求 8：系统能显示高清宽屏 480 逐行扫描节目内容（DVD 节目），分辨率达到宽屏  $852 \times 480$ 。

需求 9：系统能显示正常 480 线以上的节目内容。

需求 10：系统可连接到环绕声音响系统。

需求 11：系统提供输出连接，可以连接记录设备，如高清-DVD 录像机。

需求 12：系统保修时间不少于 10 000 个小时。

需求 13：系统必须考虑到 2020 年时高清节目内容的普及，为市场作好准备。

需求 14：系统应遵守版权协议并保留相应的法律权利。

6. 画出下列博客浏览过程的活动图：

（1）作者提交内容；

（2）编辑浏览内容；

（3）如果编辑同意，则内容将被发布到站点，否则给作者发送拒绝通知。

7. 描述你的电脑硬件，画出类图，并描述硬件部件分类及相互关系。

① PC 硬件部件：机箱，键盘，康宝 DVD 刻录机，主板，显示屏，鼠标。

② 部件之间的关系：机箱包含主板，机箱包含康宝 DVD 刻录机，机箱连接一个键盘、一个屏幕和一鼠标。

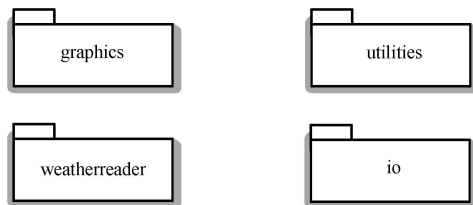
8. 组件图习题

（1）假设你在写一个股票交易软件，想对有价证券管理组件进行建模，该组件有两个已经提供的接口：TradeExecution 和 TradingHistory。一个要求的接口：IStockDAO。画出组件图，表示有价证券管理组件及其接口。

（2）修改画出的图，将 IStockDAO 接口改为通过使用持久性组件进行连接。

(3) 以画出的图为例，表示有价证券管理组件依赖于另一个组件（StockPersistenceMgr），该组件处理对数据库的持久性。

9. 假设编写一个程序，用来表示当前的天气，包含以下包：



修改包图并表示以下依赖：

- ① graphics 依赖于 Utilities；
- ② io 依赖于 Utilities；
- ③ weatherreader 依赖于 io。

10. 假设你在写一个数字宠物的程序，在不同的状态下，这个数字宠物接受到不同的刺激会有不同的改变，因此需要使用状态图对该数字宠物进行建模。数字宠物程序的行为如下：

- ① 当打开数字宠物的开关时，它开始高兴；
- ② 如果宠物高兴，接受到惩罚，那它会变悲伤；
- ③ 如果宠物悲伤并受到表扬，那它会变高兴；
- ④ 如果宠物悲伤并受到惩罚，那它就死翘翘。

识别数字宠物的状态和转换，并画出状态图。

11. 设有某应用系统，该系统向用户提供查阅论文的功能，但用户真正能使用该功能之前必须登录；此外，该查阅论文功能还提供论文下载服务（要收一定费用）；用户可以通过冲值卡或手机两种方式进行缴费，缴费记录将通过记账系统。系统管理员可向系统添加论文，添加了论文后应该修改目录。

请用用例图来表达上述功能需求。

12. 请按 P232 所要求的增加功能，建立商场销售管理系统的 OOA 模型。

## 第7章 系统实施

系统实施是将系统的设计方案转换成实际运行的系统的全过程。经过系统分析和系统设计阶段，已经得到了有关系统的全部设计信息，接下来的工作就是将文档中的逻辑系统变成真正能够运行的物理系统了。

系统实施分为两个阶段。第一个阶段是系统技术实现过程和对这个过程的管理，包括选择合适的开发平台、建立编程标准、系统编程、测试、建构和发行，这都是交付前的工作。这个阶段的交付物包括软件、数据和文档资料，最终运行的软件是交付物的核心，用户手册等其他交付物也必不可少。第二个阶段是用户转化阶段，即系统发行后交付用户使用的过程，包括用户培训、业务流程重组实施、系统转换、运行和维护。这是系统实施的用户化过程，这一阶段的交付物主要是用户实施方案，包括培训方案、重组实施方案、转换方案、运行和维护方案、维护记录与修改报告等。第一个阶段在开发团队完成，它着重于技术实现，完成的系统覆盖需求规格，达到系统目标和指标，即从技术角度实现系统，满足用户需要；第二个阶段着重于管理，在用户端完成。虽然侧重点不同，但目标都是为了系统的成功实施，给用户一个好系统，让用户用好这个系统。

### 7.1 概述

系统实施是继系统规划、系统分析、系统设计之后的又一重要阶段，它将按照系统设计选定的方案具体实现。在此期间，将投入大量的人力、物力，占用较长的时间，使用部门将发生组织机构、人员、设备、工作方法和工作流程的较大变革。信息系统的规模越大，实施阶段的任务就越复杂。为此，在系统实施开始之前，就要制定周密的计划，即确定系统实施的方法、步骤、所需的时间和费用，并监督计划的执行，以保证系统实施工作的顺利进行。

#### (1) 前期准备工作

系统实施阶段的人、财、物、技术等都要相对集中，而且系统实施阶段对组织机构的影响也非常直接，因此必须做好系统实施前的准备工作。系统分析阶段的系统分析报告和系统设计阶段的系统设计报告是系统实施的最基本的依据。在系统实施之前，项目负责人必须对之有较深入的了解，并根据要求组织好有关的准备工作。

① 制订系统的实施计划。根据系统设计的要求制订系统实现的具体计划，包括机房整装、网络建设、硬/软件安装、系统编程、系统的调试与转换等方面的计划。

② 组织好系统的实施队伍。在系统实施阶段，参加人员较多，需要适当调整和健全组织机构，加强组织管理与控制工作；要做到人员职责分工明确，各方面工作情况的信息及时反馈到项目负责人处，以及时发现问题，纠正偏差。

③ 软、硬件与配套设施的准备。在系统的总体规划或项目级的系统分析阶段，已对设备配置作了计划和安排，系统对设备的总体需求已经清楚，除了个别的设备（如某些特殊要求的输入、输出设备）需在系统设计之后才能确定之外，大多数设备在完成系统分析之后就可以进行选购。因此，到了系统实施阶段，就应当做好有关工作场所、机房、通信等设施的准备工作，并且进一步做好系统设备的采购安装和调试工作。

④ 信息流程的重组和组织业务规程修订。为了适应新系统的要求，可能需要对现行系统的信息流程进行重新组织，并相应修订原有的业务规程和工作制度，以适应新的变化。这些工作都必须使之与用户原有的工作方式不发生严重的对立性冲突，要使用户能够乐于接受新的系统。这些工作实际上是系统实施的一个重要方面。

⑤ 人员培训与宣传教育工作。人员培训包括对系统实施人员的培训和对用户的培训，对系统实施人员的培训，首先要使他们统一思想，建立共同语言，并确定统一的实施原则，使各部分能够相互协调地进行。用户培训与宣传教育是为了使用户了解系统是怎么样的，特别是要了解系统将会带来什么样的变化，如何去接受这种新的变化等。人员培训要随着系统的实施一直进行，直到用户学会操作和维护新系统为止。

## （2）系统的编制与调试

根据系统需求、系统投资及原有遗留系统的架构等各种综合因素来决定和选择系统的开发平台及开发工具，对系统进行编程，对已实现的系统进行全面的软件测试，排除一些设计中的错误和不完善的地方，并予以纠正。

## （3）数据准备

数据的收集、整理和录入是一项既烦琐、劳动量又大的工作。而若没有一定基础的数据的准备，系统的调试和测试就不能很好地进行。一般来说，确定数据库的物理模型之后，就应当进行数据的整理、录入。这样既分散了工作量，又可以为系统调试和测试提供真实的数据。

## （4）系统调试与试运行

程序设计完成后，要进行全面的系统调试。通过调试后，将现场数据装载到系统中，对系统进行试运行，对不符合用户实际要求的地方进行局部调整，同时要编写相应的技术手册及用户手册，制定系统的管理制度及操作制度，对系统操作与维护人员、终端用户及管理人员进行进一步的培训。



### (5) 系统转换

将旧的信息系统转换成新的信息系统时,可以直接切换,也可以并轨运行,即旧系统与新系统同时运行比较,直到新系统能够替代旧系统为止。转换过程也可以采用分批方式转换,将成熟的部分分批投入实际运行,直至新系统全部代替旧系统为止。在此过程中,还要进行系统的评价与验收。

系统实施后要保障系统的正常运行,即进行维护与支持。要制定系统运行维护规范,对系统的运行进行日常管理,记录每天的运行状况。同时,还要进行程序、数据文件及代码的维护等工作,处理一些软、硬件故障,完成对设备的维护与管理,估算系统的效益等。

系统实施阶段的输出文档包括程序文档和系统实施报告两大类。程序文档是今后系统维护、修改和扩充的主要的详细技术依据,主要包括程序设计报告、源程序清单及程序调试报告等。程序设计报告的主要内容包括对原设计的修改和补充。因为有少数问题直到编码和调试时才发现,这时应修改和补充模块的有关文档,乃至系统设计的有关文档。另外,程序编制应该按照国家颁布的软件设计规范设计和建立文档。

系统实施阶段的文档是系统验收、审计、评价及运行维护的依据,主要有:

- ① 系统实施计划;
- ② 设备采购及安装验收报告;
- ③ 业务规程及有关制度;
- ④ 系统调试及试运行情况报告;
- ⑤ 系统转换及验收报告;
- ⑥ 系统的操作使用手册等。

## 7.2 开发平台

开发平台的选择是由系统需求、系统投资及原有遗留系统的架构等各种因素综合决定的。J2EE和.NET是目前两大主流开发平台,二者都针对多层分布式应用系统的设计、集成、性能、安全性和可靠性等诸多方面为用户提供了总体的指南和规范,并提供了相应的工具和编程环境。

J2EE(Java 2 Enterprise Edition)是SUN公司于1996年推出的关于开发、部署、运行和管理的基于Java的分布式应用的标准平台,为基于多层分布式应用模型上的Java应用的设计、开发、装配和部署提供了一个完整的框架。

.NET是Microsoft公司于2000年6月推出的新一代Internet上的网络平台,更详细的解释应该是,.NET是一种分布式的运算框架,平台以XML为基础,以Web为核心,并结合其他多种技术,最大限度地利用Internet上丰富的资源来提高工作效率。

### 7.2.1 J2EE 平台

J2EE 平台是一种利用 Java 来简化企业应用程序的开发、部署和管理的体系结构，其技术基础是 Java 平台的标准版。J2EE 平台不仅巩固了标准版中的许多优点，如“编写一次、到处运行”的特性、方便存取数据库的 JDBC API、CORBA 技术及能够在 Internet 应用中保护数据的安全模式等，同时还提供了对 EJB（Enterprise JavaBeans）、Servlet、JSP 及 XML 技术的全面支持。

SUN 公司设计 J2EE 的部分起因是要解决 C/S 结构的缺陷。J2EE 定义了一套标准化的组件，并为这些组件提供了完整的服务。组件技术是目前软件界很流行的一种软件重用技术。在 J2EE 平台上的企业应用系统实际上是许多实现企业业务逻辑和用户界面的 J2EE 组件的集合，J2EE 平台支持的 Java 组件类型有 Applets、Application Client、EJB 和 Web 组件，这些组件在各自的容器中运行。容器是 J2EE 平台提供的系统级软件，为在它们内部运行的组件提供诸如生命管理、安全控制、事务管理及线程管理等运行时服务。组件在被组装、配置到各自容器内是通过配置文件来描述的，由容器自动实现，不需要通过编程实现。这种机制减轻了编程人员的负担，使得他们可以致力于业务逻辑的实现而不必考虑由于分布性带来的与业务无关的复杂的技术细节问题，降低了开发分布式企业级应用系统的复杂性。

Java 语言被推出时，其主要用途是制作产生动态网页的 Applet。后来发现，Java 的一次开发多次运行、纯面向对象的特性、垃圾回收机制和内置的安全性特别适合于开发企业级应用系统。于是企业开发商纷纷在 Java 标准版的基础上各自扩展出许多企业应用 API，其结果导致基于 Java 的企业应用呈爆炸式增长。但是，各企业系统之间又不能相互兼容。为此，SUN 公司联合 IBM、BEA Systems、Oracle 等大型企业开发商于 1998 年共同制定了一个基于 Java 组件技术的企业应用系统开发规范，该规范定义了一个多层企业信息系统的标准平台，以简化和规范企业应用系统的开发和部署。这一平台就构成了 J2EE。

#### 1. J2EE 体系结构

J2EE 平台使用多层分布式应用程序模型，根据功能的不同把应用程序逻辑划分成各个不同的组件，各个组件可安装在不同的机器上。组件的位置取决于组件本身在多层 J2EE 环境中所处的层次。图 7.1 演示了被分成不同层的两个多层 J2EE 的应用：

- 运行在客户机上的客户层组件；
- 运行在 J2EE 服务器上的 Web 层组件；

- 运行在 J2EE 服务器上的业务层组件；
- 运行在 EIS 服务器上的企业信息系统层（Enterprise Information System, EIS）组件。

图 7.1 多层结构的 J2EE 应用程序

从图 7.1 中可以看到，J2EE 应用系统既可以是三层结构，也可以是四层结构。三层结构包括客户层、J2EE 中间层和 EIS 层；四层结构包括客户层、Web 层、业务层和 EIS 层，其中业务层和 Web 层共同组成了三层结构中的 J2EE 中间层。一般来说，J2EE 应用系统经常分布于三个不同的位置，所以通常将 J2EE 应用系统的多层结构考虑为三层结构。对应的三个位置分别是客户端机器、J2EE 服务器和在后端的数据库服务器。三层结构的应用系统可以理解为在标准的两层结构中的客户端程序和后端服务中间增加了应用服务器。

#### （1）客户层（Client Tier）

客户层用来实现系统的操作界面和显示，某些客户层程序也可实现业务逻辑。客户层可分为基于 Web 的客户端和非基于 Web 的客户端两种情况。基于 Web 的客户端主要作为企业 Web 服务器的浏览器端，一般称为瘦客户端。非基于 Web 的客户端则是独立的应用程序，完成瘦客户端无法完成的任务。

#### （2）Web 层（Web Tier）

Web 层也叫表示逻辑层，它为系统提供 Web 服务，由 Web 组件组成。Web 组件包括 JSP 页面、基于 Web 的 Applets 及显示 HTML 页面的 Servlet。Web 层也可以包括一些 JavaBeans。Web 层主要用来处理客户请求，调用相应的逻辑块，并把结果以动态网页的形式返回到客户端。Web 层通常在 Web 服务器（如 Microsoft IIS、Jakarta Tomcat、IBM Webshpere）中实现。

### (3) 业务层 (Business Tier)

业务层也叫业务逻辑层、EJB 层或应用层,它由 EJB 服务器和 EJB 组件组成。一般情况下,把 Web 服务器和 EJB 服务器产品结合在一起,称为应用服务器。EJB 层用来实现企业级信息系统的业务逻辑,是企业级应用的核心。业务层中的 EJB 运行在容器中,容器解决了底层的问题,如事务处理、生命周期、状态管理、多线程安全管理、资源池等。

业务层和 Web 层一起构成了三层结构中的中间层。

### (4) EIS 层 (Enterprise Information System Tier)

企业信息系统层主要用于企业信息的存储管理,主要包括数据库系统、目录服务等。J2EE 应用程序组件经常需要通过访问企业信息系统层来获取所需的数据信息。

## 2. J2EE 核心技术

企业级应用系统的开发要面临许多困难,开发者需要花费大量的精力来完成事务管理、多线程、数据库连接池和其他一些底层处理。基于组件和平台无关的 J2EE 结构使 J2EE 应用系统变得易于编写,除了把业务逻辑封装到可重用组件中外,J2EE 服务器以容器的形式为每一个组件类型提供底层服务。

J2EE 提供的支持分布式企业应用的开发技术为构建大型分布式企业级应用提供了机制,根据其使用特点,可以把这些技术分为组件技术、服务技术和通信技术。组件技术用来创建用户接口和业务逻辑,通过组件模块化设计,用户可以在多个企业应用中重用组件,从而简化了应用编程,提高了系统开发效率;服务技术为应用组件提供服务,从而更高效地发挥组件的功能;通信技术对应用程序设计者来说是透明的,它为应用的不同部分之间提供了通信机制,而不管它们是本地的还是远程的。

### (1) 容器

容器是用来管理组件行为的一个集合工具,组件的行为、组件的生命周期、组件之间的合作依赖关系等。所有组件都在容器中运行,容器为组件提供服务。服务包括可设置服务,如安全性、事务管理、JNDI (Java Naming and Directory Interface) 寻址和远程连接等服务,也包括一些不可设置的服务,如 EJB 和 Servlet 的生命周期、数据库连接资源池、数据持久性及对 J2EE 平台 API 的访问。

J2EE 规范定义了 4 个主要的容器类。

① Applet 容器。Applet 容器就是运行在客户端机器上的 Web 浏览器和 Java 插件的组合,用于运行 Applet。Applet 容器包括对 Applet 编程模型的支持。

② 应用程序客户端容器 应用程序客户端容器管理所有 J2EE 应用的应用程序客户端组件的执行,运行标准的 Java 客户应用程序,提供访问 J2EE 服务和通信的 API。

应用程序客户端及其相关容器运行在客户机上。

③ Web 容器。Web 组件驻留在 Web 容器中，除了标准的容器服务外，Web 容器还提供网络服务、解码请求、格式化响应等功能。所有的 Web 容器必须支持 HTTP 协议，同时也可支持 HTTPS 协议。

④ EJB 容器。EJB 容器管理了一个 J2EE 应用中全部 EJB 的执行，EJB 及其相关容器运行在 J2EE 服务器中。除了标准容器服务外，EJB 容器还为 EJB 组件提供事务服务、持久性服务及访问 J2EE 服务和通信 API。

## (2) 组件技术

J2EE 应用系统由一系列的组件组合而成。组件是指可以重用的代码单元，代表着一个或者一组可以独立出来的功能模块，它随同与它相关的类和文件被装配到 J2EE 应用系统中，并与其他组件通信。

J2EE 规范定义了下列组件：客户端应用程序和 Applet，是运行在客户端的组件；Java Servlet 和 Java Server Page (JSP)，是运行在服务器端的 Web 组件；EJB (Enterprise JavaBean) 组件，是运行在服务器端的业务组件。

① 客户端组件。客户端可以直接与运行在 J2EE 服务器中的业务层的组件进行通信，也可以通过运行在 Web 层中的 JSP 页面和 Servlet 与业务层的组件进行通信。J2EE 的客户端既可以是 Web 浏览器、一个 Applet，也可以是一个 J2EE 应用程序客户端。

- Web 浏览器。Web 浏览器又称为瘦客户端，它通常只进行简单的人机交互，不执行像查询数据库、复杂的业务规则运算等操作。
- Applet。Applet 是一个用 Java 语言编写的小的客户端应用程序，运行在浏览器上的 Java 虚拟机中，通过 HTTP 协议与服务器进行通信。为了在浏览器中成功地运行 Applet，浏览器中需要 Java 插件和安全策略文件。
- J2EE 应用程序客户端。一个 J2EE 应用程序客户端运行在客户端计算机上，它使得用户可以处理比标记语言所能提供的更丰富的用户界面的任务。典型的 J2EE 应用程序客户端拥有通过 SWing 或抽象窗口工具包 (AWT) 应用程序接口 (API) 建立的图形用户界面。

J2EE 应用程序客户端可以直接访问服务器 EJB 容器内的 EJB 组件。当然，J2EE 应用程序客户端也可像 Applet 组件那样以 HTTP 连接与服务器中的 Servlet 通信。与 Applet 不同的是，J2EE 应用程序客户端一般需要在客户端进行安装。

② Web 组件。Web 组件是在 J2EE Web 容器上运行的软件程序，其功能是在 HTTP 协议上对 Web 请求 (Request) 进行响应 (Response)。所谓响应，其实是动态生成的网页。用户每次在浏览器上单击一个链接或图标，实际上是通过 Web 向服务器

发出请求，Web 服务器负责将 Web 请求传递给 Web 组件，J2EE 平台的 Web 组件对这些请求进行处理后回复给客户相应的 HTML 或 XML 文件。

Web 组件包括 Servlet 和 JSP。每个 Servlet 是一个 Web 容器（又称 Web 服务器）中的程序组件。Servlet 实质上是动态处理 HTTP 请求和生成网页的 Java 类。JSP 在 Web 容器内会被自动编译为 Servlet，编写 JSP 比编写 Servlet 程序更简洁。

③ EJB 组件。EJB 组件用于实现特定的业务逻辑，业务逻辑就是企业级应用中的数据结构和算法，在许多文献中也称为应用逻辑。EJB 组件能够从客户端或 Web 容器中接收数据并将处理过的数据传送到企业信息系统层来存储，还能够从企业信息系统层检索数据并送回到客户端。

J2EE 1.4 版本有三种 EJB 组件：会话 Bean（Session Bean），实体 Bean（Entity Bean）和消息驱动 Bean（Message-driven Bean）。一个会话 Bean 描述了与客户端的一个短暂会话，当客户端执行完成后，会话 Bean 和它的数据都将消失；实体 Bean 对应数据实体，它描述了存储在数据库表中的持久数据；消息驱动 Bean 是 EJB2.0 中新增加的类型，它结合了一个会话 Bean 和一个 Java 信息服务（JMS）监听者的功能。

### （3）服务技术

一个企业级应用系统往往需要一些基本的服务，如事务服务、命名与目录服务、消息服务等。这些服务将简化分布式企业级应用系统的建设，同时还可以更加有效地利用企业内的可用资源。J2EE 服务技术简化了应用编程，允许用户在部署时定制组件和应用。以下是其最重要的几种服务。

① 事务服务（Java Transaction Service，JTS）。J2EE 的事务服务采用了 OMG 的事务服务规范 OTS（Object Transaction Service），保证在分布式的环境下（如跨组件、跨主机、跨数据库）事务处理的 ACID（Atomicity, Consistency, Isolation, Durability）特性。J2EE 规范通过定义 JTA 向应用程序提供分布式事务处理服务，JTA 是一种用来运行事务的途径。

② 命名与目录服务（Java Naming and Directory Interface，JNDI）。JNDI 提供了命名和目录功能，它为应用提供方法来完成标准的目录操作，如把对象与属性关联起来和用属性来查找对象。利用 JNDI 服务应用系统可以存储和检索所有类型的命名的 Java 对象。

JNDI 有两个基本概念：Naming 和 Directory。Naming 是计算机系统中最基本的服务之一，它把名称（name）与对象（object）绑定起来，而用户则通过一个上下文（context）环境对象来存取所需的对象。Directory 是对 Naming 概念的扩充，让对象拥有属性（attribute）来记录额外的信息。这样，用户可以使用名称来查找对象，还可以获得该对象的属性信息，此外使用属性作为搜寻的过滤条件，还可以搜索符合条件的对象。

③ 安全服务。J2EE 平台定义的安全服务可以让开发者配置 Web 组件或 EJB 组件,使得只有被授权的用户才能访问系统资源。每个客户指派特定的角色,而每个角色只允许激活特定的方法。安全服务采用身份认证 (Authentication) 和资源授权访问 (Authorization) 两种模式来保证资源的安全。

④ 消息服务 (Java Message Service, JMS)。在企业级环境中,各种分布式组件往往不是保持持久的联系。因此需要采用一种手段来进行异步发送数据,JMS 通过使用面向消息的中间件 (Message-oriented Middleware, MOM) 为发送和接收消息提供这种机制。它允许 J2EE 应用程序建立、发送、接收和阅读消息,使得建立连接简单、可靠的和异步的分布式通信成为可能。

⑤ 数据存取服务 (Java DataBase Connectivity, JDBC)。J2EE 规范把 JDBC 作为它向应用系统提供的数据库存取服务的接口。通过 JDBC,可以从 Java 中访问关系型数据库,它提供了厂商之间的互连和在不同厂商提供的关系数据库之间进行数据访问。

#### (4) 通信技术

J2EE 作为一个分布式计算环境的规范,必然需要定义一些通信协议,用于各层、各组件之间的通信及实现与其他系统的互操作,通信技术提供了客户机与服务器间、不同服务器间的合作对象之间的通信机制。这些通信协议都是目前已经得到广泛使用的协议,任何与 J2EE 兼容的实现都必须采用、实现这些协议。这些协议包括 Internet 协议、远程对象协议 (RMI、RMI-IIOP、JavaIDL)、异步消息传送协议 (JMS 和 Java Mail) 等。

### 7.2.2 .NET 平台

.NET 是微软推出的企业级应用系统开发平台,其底层以 XML 作为数据交换的基础,以 SOAP 为通信协议,借助 XML 与平台、语言和协议无关的特点,打破了不同网络、不同应用软件和不同种类计算机设备之间的差别,使企业级应用系统能够发挥协同效应,从而提供了一个集成化和用户化的解决方案,其用户可在任何时间、任何地点及任何设备上对信息进行处理。

.NET 平台包括 5 个部分,如图 7.2 所示。

操作系统是 .NET 平台的基础,目前 .NET 平台仅可以运行在 Microsoft 提供的 Windows 操作系统中。

.NET Enterprise Servers 提供了一系列的 .NET 服务器产品,包括 Application Center 2000、BizTalk Server 2000、Commerce 2000 等。通过这些产品,可以缩短构建大型企业级应用系统的周期。



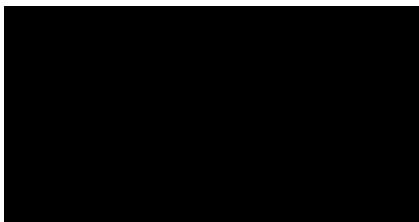


图 7.2 .NET 平台的组成

.NET Building Block Service 指的是一些成形的服务,如由 Microsoft 提供的 .NET Passport 服务。开发者可以以付费的方式直接将 these 服务集成在自己的应用程序中。

.NET Framework 位于整个 .NET 平台的中央,为 .NET 应用提供了底层的支持。事实上,即使没有位于顶层的 Visual Studio .NET,只要有了 .NET Framework,开发者一样可以开发 .NET 应用系统。

Visual Studio .NET 是 .NET 平台的集成开发环境,它位于 .NET 平台的顶端。Visual Studio .NET 集成了一系列的 .NET 开发工具,如 C#.NET、VC.NET (Visual C++.NET)、VB.NET (Visual Basic .NET) 等。

与 J2EE 相比,.NET 的开发工具更为方便一些,集成度更高一些。.NET 在推出时,吸收了许多 J2EE 平台的优点,从 .NET 开始,其应用不再以本地机器代码运行,而是编译成中间代码,由通用语言运行时 (Common Language Runtime, CLR) 中的虚拟机来运行,这样 .NET 也具备了跨平台的可能。不过 .NET 的跨平台特性主要体现在支持多种开发语言上,如 C#.NET、VC.NET、VB.NET 等,都可以被编译成相同的中间代码,使用相同的运行库执行。

### 1. .NET Framework

.NET Framework 实现了语言开发、代码编译、组件配置、对象交互、程序运行等各个层面的功能,为 Web 服务及普通应用程序提供了一个托管、安全和高效的执行环境。

.NET Framework 结构如图 7.3 所示,它包含两个主要部分:通用语言运行时 (CLR) 和具有多层次结构的统一的 .NET 基础类库。CLR 是一个软件引擎,用来加载应用程序,确认它们可以没有错误地运行,进行相应的安全许可验证,执行应用程序,然后在完成后将它们清除。.NET 基础类库,向程序员提供所需用来编写在 CLR 的控制下运行的代码的软件组件,它们按照单一有序的分级组织提供了一个庞大的功能集,



包括从文件系统到对 XML 功能的访问等。该类库为开发者提供了三种基本编程模板：基于 ASP.NET 的 Web 表单应用、基于 ASP.NET 的 Web 服务应用和基于传统 GUI 交互的 Windows 应用。



图 7.3 .NET Framework 结构

在开发技术方面，.NET 框架提供了全新的数据库访问技术 ADO.NET，以及网络应用开发技术 ASP.NET 和 Windows 编程技术 WinForms。在开发语言方面，.NET 提供了多种语言支持。

#### (1) 通用语言运行时 (CLR)

CLR 其实就是 .NET 的虚拟机，它为 .NET 应用程序提供了一个托管的代码执行环境。托管意味着将原来由程序员或操作系统做的工作剥离出来交由 CLR 来完成，从而使程序运行获得更高的安全性和稳定性。这些工作包括内存管理、即时编译、组件自描述、安全管理和代码验证，以及其他一些系统服务。

CLR 在借鉴 JVM 的自动垃圾收集、异常处理等机制的同时，又为 .NET 平台添加了多语言支持、组件自描述等新的特性。与 Java 源代码翻译成字节代码 (Byte Code) 类似，.NET 将所有 Visual Studio .NET 中性语言翻译成一种称为 Microsoft Intermediary Language (MSIL) 的中间语言而实现了 .NET 平台的跨语言承诺。运行时，中间语言被即时编译器 (JIT) 编译成特定平台的二进制代码，鉴于微软在“Wintel 平台”上的代码优化能力，.NET 代码的执行速度比 Java 有明显的优势。

CLR 根据托管组件的来源（如互联网、企业局域网、本地机器）等因素确定各组件的信任度，并根据信任度来限定它们执行诸如读取文件、修改注册表等敏感操作的权限。此外，CLR 借助通用类型系统 (Common Type System, CTS) 对代码类型进行严格的安全检查，以避免不同组件之间可能存在的类型不匹配的问题。通过代码访问

安全机制，开发人员可以为应用程序指定完成工作所必需的权限。CLR 不仅规定了代码访问安全，还规定了基于角色的安全。基于角色的认证为互联网上分布式组件的执行提供了安全保证。

CLR 通常寄宿在其他高性能服务器的应用程序中，如互联网信息服务器（IIS）、SQL Server 数据库服务器等。这样，开发者可以充分利用 CLR 诸多安全、高效的优点来部署自己的业务逻辑。

## （2）基础类库

.NET Framework 类库由一组面向对象的、可被开发者用于任何编程语言的可重用类集合组成。它提供了几乎所有应用程序都需要的公共代码，不管是传统的命令行程序还是 Windows 图形界面程序，抑或是面向下一代互联网分布式计算平台的 ASP.NET 或 Web 服务应用。

与在 Windows 其他的 SDK 中的代码库一样，.NET 基础类库将程序员从繁重的编程细节中解放出来，而专注于系统的商业逻辑。它将核心 Win32 API 最常用的功能和外挂 SDK 的功能封装到了一个统一的包中，并采用清晰而有条理的方式对类库进行分组和描述，这样开发者就能够更方便地找到其应用程序所需要的大多数功能。

## 2. .NET 核心技术

与 J2EE 有着令人眼花缭乱的各种技术名目不同，.NET 框架的一切奥妙技术都在它的 CLR 和基础类库中。下面介绍其中的托管和程序集、中间语言、通用类型系统、通用语言规范，以及对企业级 Web 应用系统比较重要的 ASP.NET 和 ADO.NET 等技术。

### （1）托管（Managed）和程序集

托管是 .NET 的一个专门概念，它是融于 CLR 中的一种新的编程理念。使用托管意味着代码可以被 CLR 所管理，使用 CLR 所提供的各种服务。也就是说，使用 .NET 托管机制就能开发出具有最新特性（如垃圾自动收集、程序间相互访问等特性）的 .NET 框架应用程序。

所有 C#.NET、VB.NET、JScript.NET 默认时都是托管的，但 VC.NET 默认时不是托管的，也就是非托管 C++，必须在编译器中使用特定选项才能产生托管代码。由托管概念所引发的托管应用程序包括托管代码、托管数据和托管类三个组成部分。

程序集作为 .NET Framework 应用程序的构造块，构成了部署、版本控制、重复使用、激活范围控制和安全权限的基本单元。程序集是为协同工作而生成的类型和资源的集合，这些类型和资源构成了一个逻辑功能单元。

程序集旨在简化应用程序部署并解决在基于组件的应用程序中可能出现的版本控制问题。最终用户和开发人员比较熟悉当今基于组件的系统所产生的版本控制和部署问题，一些最终用户经历过在计算机上安装新应用程序失败的事情，或发现已有的应用程序突然停止工作；许多开发人员花费了大量的时间来使所有必需的注册表项保持一致，以便激活 COM 类。通过在 .NET Framework 中使用程序集，这些开发问题得以解决。因为程序集是不依赖于注册表项的自述组件，所以程序集使无相互影响的应用程序的安装成为可能。程序集还使应用程序的卸载和复制得以简化。

### (2) 中间语言 (MSIL)

当托管代码被编译后，并不产生本机的二进制代码，而是产生包含中间语言 MSIL (Microsoft Intermediate Language) 的程序集。每种托管语言 (C#，VB.NET) 都产生中间语言程序集，MSIL 为被编译的代码提供了一种通用的表示。Microsoft 宣称，编译为 MSIL 的代码可以运行在任何使用 .NET Framework 的处理器和操作系统中。

被编译为 MSIL 语言的程序集不能直接运行，CLR 中使用 JIT (Just In Time) 编译器将 MSIL 转化为本机的 CPU 指令。

### (3) 通用类型系统 (CTS)

因为 .NET 统一地对待所有的语言，所以在 .NET 中要求使用 C# 书写的类与使用 VB.NET 书写的类能够互相使用，使用托管 C++ 和托管 COBOL 编写的接口也要完全一样。所有的语言必须拥有共同的标准才能够顺利地集成在一起，因此 Microsoft 定义了通用类型系统 CTS 来规范每一种 .NET 语言。换言之，CTS 定义了一组语言编译器必须遵循的规则，以定义、引用、使用和存储引用类型和值类型，这样遵循 CTS 不同语言中编写的对象才能彼此交互。有人认为，CTS 是 CLR 中最核心的部分，因为如果没有 CTS 存在，.NET 的语言的无关性就得不到体现。

CTS 执行以下功能：

- 建立一个支持跨语言集成、类型安全和高性能代码执行的框架；
- 提供一个支持完整实现多种编程语言的面向对象的模型；
- 定义各种语言必须遵守的规则，有助于确保用不同语言编写的对象能够交互作用。

### (4) 通用语言规范 (CLS)

每种语言都有自己的特性，例如，有些语言 (C++) 对大小写字母是敏感的，而 VB 则不区分大小写。为了实现语言级的继承，Microsoft 制定了通用语言规范 CLS (Common Language Specification)。CLS 提供了可以作为 .NET 语言的最小特性集，任何 .NET 语言都必须支持这些特性，它描述了一组基本的语言功能并定义了如何使用

用这些功能的规则。CLS 规则定义了通用类型系统 CTS 的子集,即所有适用于通用类型系统的规则都适用于 CLS,除非 CLS 中定义了更严格的规则。CLS 通过定义一组开发人员可以确信在多种语言中都可用的功能来增强和确保语言的互用性。

CLR 提供了最广泛的语言特性,仅有极个别的语言可以完全实现这些特性,如 Microsoft 的中间语言 MSIL。一般的 .NET 语言都是 CLR/CTS 的子集,这些语言必须包含一个共同子集 CLS——也是 .NET 语言的最小特性集。如果需要使用多种语言混合开发,那么程序员就需要遵守 CLS 以保证书写的程序的互操作性。

#### (5) ASP.NET

ASP.NET 是一种用于创建基于 Web 的应用程序的编程模型。从本质上来说,CLR 和 .NET 框架类库集可以用于创建动态 Web 页。它需要在 Web 服务器的环境中运行,如 Microsoft 的 IIS,并且根据服务浏览器请求指示在服务器上执行程序。与直接由 Web 服务器提供的静态 HTML 不同的是,ASP.NET 页面其实是在服务器上执行以后产生结果的,页面的最后生成是由许多不同的指令和数据源构造的。

ASP.NET 页面以 .aspx 扩展名存储,页面由开发人员将文本、HTML 标记及 ASP.NET 特定服务器标记和脚本组合在一起,然后存储在 Web 服务器上。当该页面被请求浏览时,服务器端程序将会用纯标记来创建一个客户端浏览器可以读懂并能够呈现的页面。因为呈现的是纯标记的格式,所以任何浏览器都能够读懂,而所有的动态过程都发生在服务器端。

由于 ASP.NET 是 .NET 框架的一部分,所以 ASP.NET 可以利用这个框架提供的所有服务,包括联网、数据访问、安全及更多其他的服务。也正因为可以使用所有这些服务,所以相比以前,能够创建更加丰富的 Web 应用程序。

与其他同类产品(如 ASP、JSP)相比,ASP.NET 有其鲜明的特点。

① 语言独立性。因为 ASP.NET 是 .NET 框架的一部分,所以可以使用自己熟悉的 .NET 语言来构建 ASP.NET 应用程序,如 C#、VB 或 J# 等。其中,C# 由于其优良的语言特性而最被广泛使用。

② 事件驱动编程模式。在典型的 Web 开发中,页面总是以自顶向下的线性方式执行的,并且 HTML 标记常常与程序指令混合在一起,这样会使页面难以阅读,同时也难以维护。ASP.NET 引入了 Windows 开发者非常熟悉的事件驱动模型,这个模型允许开发者将代码与标记内容分离,将代码并入处理专门任务的有意义的单元中,如响应客户端的按钮单击动作等。事件驱动编程模式的引入,极大地提高了页面的可读性和可维护性。

③ 服务器控件。基于 ASP.NET 的 Web 编程的一个最大的特点,就是将公共的呈现和行为封装成服务器控件,从而可以在应用程序中很方便地重复使用,这样可

以帮助开发人员极大地提高开发效率。ASP.NET 包含了大约 80 多个服务器控件, 这些控件封装了从标准的表单元素到复杂控件(如网格和菜单)的几乎所有的 Web 内容。

#### (6) ADO.NET

ADO.NET 是继 ODBC 与 ADO 之后, Microsoft 特别为 .NET 框架设计的数据访问层, 也是 Microsoft 所主推的存取数据的最新技术。ADO.NET 为创建分布式数据共享应用程序提供了一组丰富的组件, 它提供了对关系数据、XML 和应用程序数据的访问, 因此是 .NET Framework 中不可缺少的一部分。ADO.NET 支持多种开发需求, 包括创建由应用程序、工具、语言或 Internet 浏览器使用的前端数据库客户端和中间层业务对象。

相比之前的 ADO, ADO.NET 提供了两种数据访问的模式: 一种为连接模式, 另一种为非连接模式。非连接模式是 ADO.NET 中才具有的, 在非连接模式下, 一旦应用程序从数据源中获得所需的数据, 它就断开与数据源的连接, 并将获得的数据以 XML 的形式存放在内存中, 在应用程序处理完数据后, 它再取得与原数据源的连接并完成数据的更新工作。

### 7.2.3 J2EE 和 .NET 平台的异同

J2EE 和 .NET 在目标和体系结构上极其相似, 但在实现上又各不相同。这两个平台中都包含了一系列的技术, 通过这些技术可以缩短开发周期, 提高开发效率, 节约应用系统的构造成本。同时这两个平台都在安全性、扩展性和性能方面做出了努力, 都提供了一系列的技术可供选择。因为这两个平台要解决的问题类似, 所以很多技术也非常类似, 有些概念设置仅仅是名称上的差别而已, 如 J2EE 中包的概念和 .NET 中名字空间的概念。两个平台的类似之处远远多于相异之处。下面从平台构造、可移植性及开发过程的角度来对两者进行比较, 通过对比加深读者对 J2EE 和 .NET 平台的理解, 以能够在工作中根据实际需要确定选用的平台和技术。

#### 1. 平台构造

一个平台在语言编译、代码执行、编程支持等基础构造方面往往会对其可用性、生产性、移植性等产生重要的影响, 也是一个平台是否适用的重要依据。J2EE 和 .NET 两个平台在底层的执行引擎都源于托管的虚拟机概念, .NET 的 CLR 在借鉴了 J2EE 的 JVM 的自动垃圾收集、异常处理等机制的同时, 又为 .NET 平台添加了多语言支持、组件自描述等新的特性。

在 .NET 和 J2EE 平台上，程序的编译都经过两个类似的过程。首先，特定高级语言编译器将 C#（及其他 .NET 语言）和 Java 源代码分别翻译成中间语言（MSIL）和字节代码。.NET 在中间语言设计时通盘考虑了多个主流高级语言，在这一层面实现了 .NET 平台的跨语言承诺。J2EE 的基石是 Java 语言，它最典型的特征是一次编写、多次运行，跨平台是 J2EE 一直引以为豪的关键，这是通过 JVM 来实现的。

其次，在运行时，.NET 的中间语言被即时编译器（JIT）编译成特定平台的二进制代码，J2EE 的字节代码则通过 JVM 解释执行，完成各自语言的指令功能。鉴于微软在 Windows 平台上的代码优化功底，所以在 Windows 平台上，.NET 代码的执行速度较之于 Java 有明显的优势是不争的事实。在 Unix/Linux 平台上，由于 .NET 迟迟未能实现其跨平台的承诺，所以 J2EE 几乎成了惟一的选择，执行效率的比较也就无所谓了。在代码执行的同时，CLR 和 JVM 也都实现了异常捕捉、类型安全、内存分配、垃圾收集等自动化内存管理工作，大大减轻了企业级信息系统的内存泄漏问题和程序员繁重的负担。

## 2. 可移植性

在可移植性方面，我们从操作系统平台可移植性和编程语言可移植性两方面来衡量。

### （1）操作系统平台可移植性

J2EE 通过 Java 虚拟机来消除操作系统平台差别，它继承了 Java 平台无关性的优点。在任何平台上，只要有 Java 虚拟机就能在不同平台上执行同一个 Java 程序。J2EE 的平台无关性使得任何符合 J2EE 标准的应用服务器之间可以共用标准的组件，各个应用组件之间也能很好地进行互操作。利用 Java 的跨平台特性，J2EE 组件可以很方便地移植到不同的应用服务器环境中。J2EE 应用服务器在 Windows 平台上也可很好地工作，这是 J2EE 的一大亮点，也是在选择企业级应用系统开发平台时的一个重要参考因素。

几乎所有的主流操作系统都提供了对 J2EE 的支持，所以实际上如果要搭建基于 Unix、Linux、Solaris 等非 Windows 操作系统平台的应用开发，或者要搭建跨多个不同操作系统平台的应用开发，J2EE 平台几乎是惟一的选择。

而出于推广 Windows 平台的商业目的的考虑，跨平台是 Microsoft 所不想的。选择 .NET 平台就意味着选择 Windows，这句话至少在可预见的一段时间里仍然是一个基本事实。

### （2）编程语言可移植性

现在的软件开发环境中，编程语言百家争鸣，软件开发人员也是各有所长，这往



往给软件系统的集成工作带来了不便，导致了企业级应用系统往往开发厂家不同，应用技术各异，版本不尽相同，使得系统集成人员在进行系统集成时面临各种各样复杂的问题。开发语言多样性的问题，对开发平台在开发语言多样性方面提出了更高的要求。

.NET 能够支持多种开发语言，如 VB.NET、VC.NET、C#、Fortran 等十几种开发语言，同时 .NET 还能够支持各种潜在的语言，只要该语言的子集/超集已被定义，并且为它们建立了 IL 编译器。但 .NET 对 Java 的支持不是很好，这是由客观的竞争环境所限定的。

这样一来，.NET 的一项很大的优势就在于它可以使应用程序的开发者不必进行新语言的学习就能够从原有的旧开发环境转到一个全新的开发环境中去，用“跨语言”的交互性的长处来平衡“跨平台”方面的不足。

相比之下，J2EE 在支持多种开发语言方面的表现不很理想。J2EE 完全是用 Java 编写的，是一个完全的单一语言的平台。任何应用程序的开发者，如果要用 J2EE 进行应用开发的话，则他必须从基础开始学习一种全新的面向对象的开发语言——Java。

### 3. 开发过程

一个平台开发与部署的便捷与否、开发周期的长与短，很大程度上会左右开发人员的选择。

#### (1) 开发方式

任何一个第一次使用 .NET 进行开发的人，都会对它的友好性欣喜不已。强大的 IDE 工具 Visual Studio .NET 无缝地结合了数据库，使得原先抽象的数据库编程在 .NET 里面非常直观；它也提供了大量的优秀的控件，如 GridView、DataSource 等，以避免用户不停地“重复开发轮子”；还可以根据自己对控件的需要，用面向对象的方式来实现更合适的控件；它还吸收了 Delphi 的所见即所得的开发方式，支持拖拽控件到设计版面上来构成一个合格的产品。不过，.NET 在把编程简单化的同时，把太多的内部机制隐藏了起来，这在一定程度上阻碍了开发人员对系统开发的整体把握，因为很多东西对 .NET 开发人员来说都是神秘的。

J2EE 平台从框架到具体应用到 IDE 工具都有开源的选择，这为开发省下了一笔开销。与 .NET 相比，J2EE 开发过程中缺少大量控件的支持，导致程序员们重复地进行“开发轮子”的活动，这不得不说是 J2EE 的软肋所在。不过也正是这样的开发方式，能够使开发人员从本质上理解 J2EE 大型应用系统的开发。

在 J2EE 的开发中，“配置”是一个很常见而且重要的过程。大量的框架都是通过相应的配置文件啮合起来的，如 Struts 的 struts-config.xml、Hibernate 的 hiber-

nate.cfg.xml 和 EAR 包里面的用于部署的必不可少的 application.xml 等。在 J2EE 里，环境配置正确是产品正常可用的前提，相比之下在 .NET 开发中配置的工作量要小得多。

### （2）开发周期

效率一直是微软所追求的，Visual Studio .NET 不仅提供了一个非常优秀的开发环境，并且对调试的支持非常好，它允许生产线式的调试，伴随着代码执行可以从一个结果到另一个结果。

J2EE 的开发工具虽然有很多，如 SUN 的 Forte、Borland 的 JBuilder 和 IBM 的 RAD，但与微软 .NET 的开发平台相比，在数据库的集成性与易用性方面还有一定的差距。这是由于不同厂商的数据库、Web 服务器、中间件服务器等都有些细微的差别所致，要开发真正的跨平台产品就需要对所有的产品都精通，分别为不同的搭配设置不同的参数，而且要进行长时间的调试。而不像 .NET，所有的前台、后台，甚至于操作系统平台都是微软的产品，自然可以整合得更加天衣无缝。

因此，如果开发同一个项目，使用 J2EE 的进度可能会比使用 .NET 慢一些，而且 .NET 的整个平台、开发工具的高集成性和友好的开发环境给开发工作带来更多的便利。

### （3）部署方式

.NET 中有一个著名的名词，叫做 XCOPY。它使得 .NET 程序集的部署与以前版本相比显得简单得多。XCOPY 部署意味着在很多情况下，都只须简单地将 .NET 应用程序目录复制到目标位置即可。

J2EE 平台的特色之一在于开发人员可以在其之上整合不同的组件，这个将组件整合为模块并将模块整合为应用程序的过程叫做打包。而在一个可使用环境的安装和定制应用程序的过程则叫做部署。J2EE 程序的部署就是将应用组件按照一定的格式打包并放置在应用服务器的容器内。J2EE 平台为打包和部署提供了相应的工具，它使用 Java 档案文件（JAR）作为组件和应用程序打包之后的标准整合结果。相对于 .NET 的 XCOPY 模式部署而言，J2EE 平台的部署复杂得多，可借助专门的部署工具进行部署。

## 7.3 系统编程

### 7.3.1 编程概述

所谓编程，就是将软件详细设计的结果翻译成用某种程序设计语言书写的程序。



软件的设计开发过程经过需求分析、总体设计和详细设计几个阶段，已经形成了基本编程框架，最后就是在选定的开发平台上通过编程对设计进一步具体化，实现相应的功能。编程的质量也是影响系统质量的十分重要的因素，如果编程中存在各种问题，那么再好的设计也无法体现出来。另外，编程质量的好坏，也直接影响软件测试和软件维护工作的进行。影响编程质量的因素是多方面的，选择的编程语言、采用的编程风格及开发队伍的整体素质等都会对编程质量产生影响。

7.3.2 编程语言

编程语言将在不同程度上影响生产效率和代码质量。不同的项目，不同的开发团队可能选择不同的编程语言。一个项目中有时会选择多种编程语言。编程语言可分为以下 3 类。

(1) 语言符号类，使用与语言类似的文本来构建软件。这些文本遵循一定的语法规则，并提供相应的语义，使用者可以直观地理解软件将完成的工作，如 HTML、XML 等标记语言。

(2) 形式化符号类，采用精确的、形式化的定义的符号来构建软件。形式化符号和形式化方法是系统级编程的重要形式，如 Pascal 语言。

(3) 可视化符号类，采用可视化的部件来构建软件，通常用于界面可视化符号元素的编程，如 Visual Basic 语言。

表 7.1 所示是对一些常用编程语言的使用概率进行排名的情况。

表 7.1 常用编程语言的使用概率排名表

排名 2008.12	排名 2007.12	编程语言	比例 (%)	排名 2008.12	排名 2007.12	编程语言	比例 (%)
1	1	Java	19.367	11	9	Ruby	2.308
2	2	C	16.163	12	12	D	1.185
3	5	C++	10.893	13	13	PL/SQL	1.140
4	4	PHP	9.479	14	14	SAS	0.843
5	3	(Visual) Basic	9.478	15	19	Pascal	0.689
6	8	C#	4.643	16	15	COBOL	0.631
7	6	Python	4.567	17	16	ABAP	0.603
8	7	Perl	3.603	18	21	Logo	0.569
9	10	JavaScript	3.062	19	17	Lisp/Scheme	0.515
10	11	Delphi	3.055	20	20	Lua	0.494

(数据来源于 TIOBE 2008 年 12 月统计)

### 7.3.3 编程风格

编程风格又叫编码风格或程序设计风格。随着系统规模和复杂性的加大，系统源程序不仅仅要被计算机编译执行，还要经常被人阅读。例如，在设计测试用例、查找错误、改正错误时源程序都要由程序作者或其他人员阅读。因此，程序的可读性变得非常重要。编程风格指的是在编程时应该遵守的一些原则。遵守这些原则可以使程序更容易被阅读和修改。

一个具有良好风格的程序应具有如下特点。

① 易于测试和维护。

② 易于修改。

③ 设计简单。为了使程序易于理解、测试和维护，最好的办法是使程序设计简单。坚决摒弃炫耀编程技巧及把程序设计复杂化的任何想法和做法。

④ 高效。对程序中效率要求较高的模块通过改进算法、数据结构和程序的逻辑结构来提高程序的效率。

一个逻辑正确但杂乱无章的程序是没有什么价值的，因为它无法供人阅读，难于测试和维护。因此应该养成良好的编程风格。良好的编程风格包含程序内部文档风格和标识符命名风格两个方面。

#### 1. 程序内部文档风格

一个程序不仅要有外部文档（如概要设计和详细设计文档），而且程序内部也应该有完整的文档。完整一致的内部文档是帮助读者理解程序的重要手段。程序的内部文档包括程序的注解和程序的书写格式两个方面。

##### （1）程序的注解

几乎所有的程序设计语言都提供了注解语句，允许程序员对程序进行说明。程序注解主要有两种类型：说明性注解和功能性注解。

① 说明性注解出现在模块的开始位置，一般包括：

- 模块的名称；
- 模块的功能和性能；
- 调用格式；
- 界面描述，包括上级调用模块、本模块调用的下级模块、输入/输出参数的含义和类型及该模块所引用的全局变量等；
- 开发历史，如作者、审查者、日期、修改的日期和修改的描述等内容。

如果一个模块的规模较大，其中包含多个函数或子程序，则可以在模块的开始增加目录性注解，说明模块中的函数或子程序的位置和功能。

② 功能性注解是指在程序中每个具有独立功能的程序段之前说明该程序段功能的注解。书写功能性注解应该注意以下几个方面：

- 应描述独立功能的程序块，而不是对每个语句加以说明；
- 注解不应是程序语句的重复，而应起补充说明作用；
- 应使用注解符和空行以便与程序段区分；
- 注解应与代码一致。修改程序时应相应修改程序中对程序段的注解，因为与程序功能不一致的注解在其他人阅读程序时会引起误解。

## (2) 程序的书写格式

程序代码的书写格式对于程序的可读性也有很大的影响。不同的程序设计语言对程序的书写格式要求不一，一些程序设计语言对程序书写格式要求较严格，而有一些程序设计语言的书写格式比较随意。例如，C 语言程序语句的书写格式比 FORTRAN 语言随意得多。格式凌乱的程序的可读性将大大降低。通常，在项目开发过程中应该对程序的书写格式进行规范，以便程序员之间相互协作交流。

程序的书写格式应有助于读者理解程序，一般要注意以下几点。

① 不要一行书写多条语句，这样会掩盖程序的逻辑结构。虽然现代大部分的程序设计语言都允许在一行中书写多条语句，但这样会使程序的结构变得不清楚。

② 用缩排格式限定语句群的边界。缩排格式要显示程序的逻辑结构。常见的一些控制结构都应该使用缩排格式，如循环语句、条件语句等控制结构的内层语句，应退格书写。缩排格式可参考具体的编程语言。

③ 在程序段之间、程序段与注解之间用空行和注解符来分割。有时也可以借助一些自动工具来实现一致的程序格式。常见的一些格式化工具通常可以对程序中的注解及控制语句的缩排形式进行规范。

## 2. 标识符命名风格

标识符的命名是程序风格的重要内容，标识符包括变量名、函数名、子程序名、文件名等。标识符的选择不应该仅仅是满足程序设计语言的语法限制，好的标识符对程序的可读性有很大的影响。变量是程序设计中用得最多的标识符之一，下面主要讨论变量命名的风格。

### (1) 变量命名的一般原则

初学程序设计语言的人往往习惯使用一些比较随意的变量名，如  $x$ 、 $y$ 、 $z$  等，当程序规模较大时，这一类变量看起来很混乱，从变量名上难以判断变量的类型和作用，

而且往往会出现很多相似的变量名，使程序的可读性降低。软件开发规范要求变量命名应该做到以下几点。

① 使用有意义的变量名。变量命名应能反映变量的意义和含义，以使它能正确地提示该程序对象所代表的实体，并能帮助读者理解和记忆。

② 使用不易混淆的变量名。过于相似的变量名，容易引起输入错误和误解。

③ 同一变量名不要具有多种含义。这种情况使读者在阅读程序时易于误解，也不便于修改。

④ 显式说明一切变量。为了易于理解和避免出错，所有变量都应该显式说明后再使用。有些程序设计语言允许对变量名不进行说明就直接使用，如 FORTRAN 语言、BASIC 语言等。在使用这些语言编程时应特别注意，由于变量可以不定义而直接使用，所以在输入源程序时如果变量名输入错误，编译器也不能检测。其他一些高级语言，如 C、PASCAL 等程序设计语言不允许使用未定义的变量，在编程时可避免这一类错误。

⑤ 对变量名作出注解，说明其含义。

## （2）匈牙利命名规则

匈牙利命名规则是影响较大的标识符命名规则，它是由匈牙利人 Charles Simonyi 于 1972 年发明的一种给变量取名字的方式。最初这种命名规则并没有得到足够的重视，自从微软公司在 Windows 中使用了该命名规则之后，它得到广泛的重视和应用。

# 7.4 系统测试

## 7.4.1 测试概述

在系统开发的每个过程中，面对着错综复杂的问题，人的主观认识不可能完全符合客观现实，与系统密切相关的各类人员之间的通信和配合也不可能完美无缺，因此，在系统生存周期的每个阶段都不可避免地会产生差错。我们力求在每个阶段结束之前通过严格的技术审查，尽可能早地发现并纠正差错。但是，经验表明审查并不能发现所有的差错。此外，在程序编码过程中还不可避免地会引入新的错误。为了保证新系统运行的正确性和有效性，将一切可能发生的问题和错误尽量排除在正式运行之前，需要进行系统测试。系统测试通常包括平台测试和应用软件测试，平台测试可分为硬件平台测试和网络平台测试两部分。由于信息系统开发的根本任务很大程度上是软件系统的开发，因此系统测试的主要对象是软件系统，信息系统测试通常指软件测试。

目前，系统测试仍然是保证系统质量的关键步骤，它是对系统规格说明、设计、编码和集成的最后复审，以获得对软件质量的再一次验证与确认。

## 1. 测试目标和原则

正如 E. W. Dijkstra 的名言所指出的：“测试只能证明程序有错（有缺陷），不能保证程序无错。”因此，能够发现程序缺陷的测试是成功的测试。当然，最理想的是进行程序正确性的完全证明，遗憾的是除非是极小的程序，否则至今还没有实用的技术能证明任一程序的正确性。

G. Myers 给出了关于测试的一些规则，这些规则也可以看成是测试的目标或定义：

- ① 测试是为了发现程序中的错误而执行程序的过程；
- ② 好的测试方案是可能发现迄今为止尚未发现的错误的测试方案；
- ③ 成功的测试是发现了迄今为止尚未发现的错误的测试。

从上述规则可以看出，测试的含义是“为了发现程序中的错误而执行程序的过程”。这和某些“测试是为了表明程序是正确的”、“成功的测试是没有发现错误的测试”等观点是完全相反的。正确认识测试的目标是十分重要的，测试目标决定了测试方案的设计。如果为了表明程序是正确的而进行测试，就会设计出一些不易暴露错误的测试方案；相反，如果测试是为了发现程序中的错误，就会力求设计出最能暴露错误的测试方案。

测试的目标是为了发现系统中隐藏的错误，应该说，这是一个非常清楚的目标。可是，由于受开发机构的利益、开发者个人的心理状态等诸多因素的影响，测试目标往往被人误解或误用。例如，一些开发机构可能希望通过测试而向用户证实系统中不存在错误，以此来表明系统能够完全满足用户的要求，于是开发机构在进行系统测试时，就可能产生回避系统错误的意图。他们往往会设计一些不易暴露系统中错误的测试方案，选择那些难以使系统出故障的测试用例，或面对用户进行一些如同表演的所谓测试操作，等等。显然，这样的测试只会使系统错误被掩盖起来，而无益于系统质量的提高。

由于测试的目标是暴露程序中的错误，从心理学角度看，由程序的编写者自己进行测试是不恰当的。因此，在系统测试阶段通常由其他人员组成测试小组来完成测试工作。

测试阶段的基本任务应该是根据软件开发各阶段的文档资料和程序的内部结构，精心设计一组好的测试用例，利用这些测试用例执行程序，找出软件潜在的缺陷。一个好的测试用例很可能找到迄今为止尚未发现的缺陷。在设计有效的测试用例之前，软件工程师必须理解软件测试的基本原则。Davie 提出了如下一组测试原则。

- ① 所有的测试都应追溯到用户需求。正如前面所讲，软件测试的目标在于揭示错误。而最严重的错误（从用户角度看）是那些导致程序无法满足需求的错误。
- ② 应该在测试工作真正开始的较长时间之前就进行测试计划。测试计划可以在需求分析阶段完成后就开始。详细的测试用例定义可以在概要设计和详细设计确定后就立即开始。因此，所有测试可以在任何代码产生之前进行计划 and 设计。
- ③ Pareto 原则应用于软件测试。Pareto 原则暗示着测试发现的错误中的 80% 很可能起源于程序模块中的 20%，当然，问题在于如何孤立这些有疑点的模块并进行彻底的测试。
- ④ 测试应该从“小规模”开始，逐步转向“大规模”。最初的测试通常把焦点放在单个程序模块上，进一步测试的焦点则转向在集成的模块簇中寻找错误，最后在整个系统中寻找错误。
- ⑤ 穷举测试是不可能的，即使对于一个大小适度的程序，其所有的运行路径排列的数量也非常大。因此，在测试中不可能运行路径的每一种组合，正确的方法是充分覆盖程序逻辑，并确保程序设计中使用的所有条件都被测试过。
- ⑥ 为了达到最佳效果，应该由独立的第三方来构造测试。“最佳效果”指最可能发现错误的测试，创建系统的软件工程师并不是构造软件测试的最佳人选。

2. 测试中的信息流

由于测试的对象主要是软件，所以在系统测试阶段的信息流主要分析软件方面的信息流程。测试阶段的信息流如图 7.4 所示。

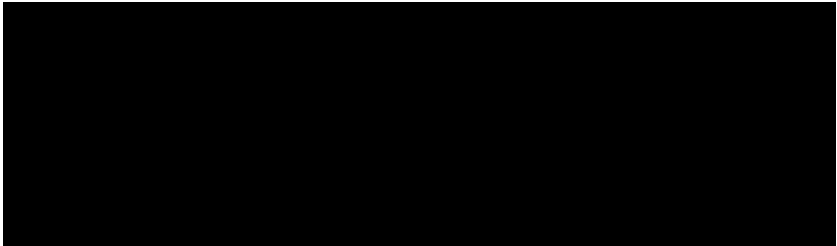


图 7.4 测试阶段的信息流

输入信息分成软件配置、测试配置和测试工具三类。软件配置由系统需求规格说明、系统设计规格说明和源程序等组成。测试配置由测试计划、测试方案组成。测试方案不仅仅是测试时使用的输入数据，还包括每组输入数据要检测的功能和预期的输出结果。测试工具不是必需的输入项，利用它来进行测试工作主要是为了提高测试效

率，实现自动化测试，测试工具有测试数据自动生成程序、静态分析程序、动态分析程序、测试结果分析程序及驱动测试的测试数据库等。

测试完成后，要对所有的测试结果进行分析，如果发现出错数据，则意味着软件有错误，就要排错并修正有关文档，接着再测试，直到通过为止。

收集和分析测试结果，进行软件可靠性分析。如果经常出现需要修改设计的严重错误，则软件质量与可靠性值得怀疑，需进一步测试。如果软件能够完成正常功能，测试中出现的错误易于修改，则一种可能性是软件的质量和可靠性基本满足要求；另一种可能性则是所做的测试尚不足以发现严重的错误。如果测试未发现错误，则应重新审查测试配置。

### 7.4.2 系统测试方法

系统的测试方法是多种多样的，可以从不同的角度加以分类：

- ① 按照是否需要执行被测系统的角度来分，可分为静态测试和动态测试；
- ② 按照测试是否针对系统的内部结构和具体实现算法的角度来分，可分为白盒测试和黑盒测试；
- ③ 按照软件测试的手段来分，有手工测试和自动测试；
- ④ 按照测试过程来分，有单元测试、集成测试、确认测试和综合测试等。

#### 1. 静态测试与动态测试

静态测试是指被测试程序不实际运行，而是采用人工检测和计算机辅助静态分析的手段对程序进行检测，主要对被测程序的编程格式、结构等方面进行评估。动态测试则是指通常意义上的测试——通过运行和使用被测试程序，发现软件故障，以达到检测的目的。

模拟这两种测试的最好方法是研究一下汽车的检查过程。踩油门、看车漆、打开前盖检查都属于静态测试技术，而发动汽车、听发动机的声音、上路行驶则属于动态测试技术。

通常，在静态测试阶段进行以下一些检测活动。

- 检查算法的逻辑正确性，确定算法是否实现了所要求的功能。
- 检查模块接口的正确性，确定形参的个数、数据类型、参数顺序是否正确，确定返回值的类型及返回值的正确性。
- 检查输入参数是否有合法性检查。如果没有合法性检查，则应确定该参数是否的确不需要合法性检查，否则应加上参数的合法性检查。经验表明，缺少参数



合法性检查的代码是造成软件系统不稳定的主要原因之一。

- 检查调用其他模块的接口是否正确，检查实参类型是否正确，实参个数是否正确，返回值是否正确，是否会误解返回值所表示的意思。
- 检查是否设置了适当的出错处理，以便在程序出错时，能对出错部分进行重做安排，保证其逻辑的正确性。
- 检查表达式、语句是否正确，是否含有二义性。对于容易产生歧义的表达式或运算符优先级，可以采用“（）”运算符以避免二义性。
- 检查常量或全局变量的使用是否正确。检查标识符的定义是否规范、一致，变量命名是否能够见名知义，简洁、规范和容易记忆。
- 检查程序风格的一致性、规范性，代码是否符合行业规范，是否所有模块的代码风格一致、规范、工整。检查代码是否可以优化，算法效率是否最高。
- 检查代码是否清晰、简洁和容易理解（注意：冗长的程序并不一定是不清晰的）。
- 检查模块内部的注释是否完整，是否正确地反映了代码的功能。错误的注释比没有注释更糟。

静态测试主要有如下3种方法。

（1）桌前检查，指程序员本人在程序通过编译后、进行单元测试之前，对源程序代码进行分析、检验，发现程序中的错误并补充相关的文档。

（2）走查。通常由3~5人组成测试小组，测试人员应是没有参加该项目开发的有经验的程序设计人员。在走查之前，应先阅读相关的文档和源程序，然后测试人员扮演计算机的角色，将一批有代表性的测试数据沿程序的逻辑走一遍，监视程序的执行情况，随时记录程序的踪迹，发现程序中的错误。由于人工检测程序很慢，因此只能选择少量简单的用例来进行，通过这种“走查”的进程来不断地发现程序中的问题。

（3）会审。其测试人员的构成与走查类似，要求测试人员在会审之前应当充分阅读有关的文档和源程序等，根据经验列出尽可能多的典型错误，然后把它们制成表格。根据这些错误清单（也叫检查表）提出一些问题，供在会审时使用。在会审时，由编程人员逐句讲解程序，测试人员逐个审查、提问，讨论可能出现的错误。实践表明，编程人员在讲解、讨论的过程中能发现自己以前没有发现的错误，使问题暴露。会审后要将发现的错误登记、分析和归类，一份交程序员，另一份妥善保管，以便再次组织会审时使用。

在代码审查时，需要注意：一是在代码审查时，必须要检查被测软件是否正确通过编译，只有在正确通过了编译后才可进行代码审查；二是在代码复审期间，一定要保证有足够的时间让测试小组对问题进行充分的讨论，只有这样才能有效地提高测试效率，避免走弯路。



经验表明,使用人工静态测试可以发现大约30%~70%的逻辑设计和编码错误。但是,代码中仍会有大量隐藏的故障无法通过静态测试发现,因此必须通过动态测试进行详细的分析。

动态测试是指在计算机上直接使用测试用例运行被测程序,检查程序的动态行为和运行结果的正确性。一般意义上的测试主要是指动态测试。根据动态测试在软件开发过程中所处的阶段和作用,动态测试可分为单元测试、集成测试、确认测试和综合测试。

## 2. 白盒测试与黑盒测试

白盒测试也称结构测试或逻辑驱动测试,它知道产品内部的工作过程,可通过测试来检测产品的内部动作是否按照规格说明书的规定正常进行;按照程序内部的结构测试程序,检验程序中的每条通路是否都能按预定的要求正确工作,而不管它的功能。白盒测试的主要方法有逻辑驱动、基本路径测试等,主要用于软件验证测试。

在理想情况下,白盒测试应该是穷举路径测试,在使用这一方案时,测试者必须检查程序的内部结构,从检查程序的逻辑着手,得出测试数据。然而贯穿程序的独立路径数是天文数字,即使每条路径都测试了,仍然可能有错误。

白盒测试主要对程序模块进行如下检查:

- ① 对程序模块的所有独立的执行路径至少测试一遍;
- ② 对所有的逻辑判定,取“真”与取“假”的两种情况都能至少测一遍;
- ③ 在循环的边界和运行的界限内执行循环体;
- ④ 测试内部数据结构的有效性。

黑盒测试也称功能测试或数据驱动测试,它是在已知产品所应具有功能的情况下,通过测试来检测每个功能是否都能正常使用的。在测试时,把程序看作一个不能打开的黑盒子,在完全不考虑程序内部结构和内部特性的情况下,测试者在程序接口进行测试,它只检查程序功能是否按照需求规格说明书的规定正常使用,程序是否能适当地接收输入数据而产生正确的输出信息,并且保持外部信息(如数据库或文件)的完整性。黑盒测试方法主要有等价类划分、边值分析、因果图、错误推测等,主要用于软件确认测试。

理想情况下,黑盒测试应该是穷举输入测试,只有把所有可能的输入都作为测试情况使用,才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个,人们不仅要测试所有合法的输入,而且还要对那些不合法但是可能的输入进行测试。

黑盒测试主要是为了发现以下几类错误：

- ① 是否有不正确或遗漏的功能？
- ② 在接口上，输入是否能正确地接受？能否输出正确的结果？
- ③ 是否有数据结构错误或外部信息（如数据文件）访问错误？
- ④ 性能上是否能够满足要求？
- ⑤ 是否有初始化或终止性错误？

表 7.2 所示是从优点、缺点和性质方面对白盒测试与黑盒测试的一个比较。

表 7.2 白盒测试与黑盒测试对比表

	黑盒测试	白盒测试
优点	适用于各测试阶段 从产品功能角度测试 容易入手生成测试数据	可以构成测试数据，使特定部分得到测试 有一定的充分性度量手段 可获得较多工具的支持
缺点	某些代码段得不到测试 如果规格说明有误则无法测试 不易进行充分性度量	不易生成测试数据 无法对未实现规格说明部分测试 工作量大，通常只用于单元测试
性质	是一种确认技术	是一种验证技术

不论采用上述哪种测试方法，只要对每一种可能的情况都进行测试，就可以得到完全正确的程序。然而，包含所有可能情况的测试（称为穷尽测试）对于实际程序而言通常是不可能做到的。为了做到穷尽测试，至少必须对所有输入数据各种可能值的排列组合都进行测试，但是，由此得到的应测试的组合数目往往大到实际上根本无法测试的程度。严格地说，这还不能算穷尽测试，为了保证测试能发现程序中的所有错误，不仅应该使用有效的输入数据，还必须使用一切可能的输入数据（如不合法的整数、实数、字符串等）。实践表明，用无效的输入数据比用有效的输入数据进行测试，往往能发现更多的错误。

因为不可能进行穷尽测试，所以测试不可能发现程序中的所有错误，也就是说，通过测试并不能证明程序是正确的，但是，测试的目的是要通过测试保证软件的可靠性。因此，为了用有限的测试发现更多的错误，需要精心设计测试用例。选择测试用例是软件测试人员最重要的一项任务。不正确的选择可能导致测试量过大或者过小，甚至测试目标不对。准确评估风险，将不可穷尽的可能性减少到可以控制的范围是成功的诀窍。

3. 验证测试与确认测试

系统包括程序及开发、使用和维护程序所需的所有文档。程序只是软件产品的一个组成部分，表现在程序中的故障，并不一定是由编码所引起的。实际上，需求分析、

设计和实施阶段都是系统故障的主要来源。因此，系统测试不仅包含对代码的测试，而且包含对系统文档和其他非执行形式的测试。

一种称之为验证的测试就是针对开发过程中的任何中间产品进行的测试。按照 IEEE/ANSI 的定义，验证测试是为确定某一开发阶段的产品是否满足在该阶段开始时提出的要求而对系统或部分系统进行评估的过程。

所谓验证，是指确定系统开发的每个阶段、每个步骤的产品是否正确无误，是否与其前面开发阶段和开发步骤的产品相一致。验证就是对诸如需求规格说明、设计规格说明和代码之类的产品进行评估、审查和检查的过程，属于静态测试。如果是针对代码，则其含义就是代码的静态测试——代码评审，而不是动态执行代码。验证测试可应用到开发早期一切可以被评审的事物上，以确保该阶段的产品正是所期望的。

另一种称之为确认的测试则只能通过运行代码来完成。按照 IEEE/ANSI 的定义，确认测试是在开发过程中或结束时，对系统或部分系统进行评估以确定其是否满足需求规格说明的过程。

所谓确认，是指确定最后的产品是否正确无误。比如，编写出的系统与用户需求和用户提出的要求是否符合，或者说输出的信息是否是用户所要求的信息，系统在整个系统的环境中能否正确稳定地运行。正式的确认包括实际系统或仿真模型的运行，确认是“基于计算机的测试”过程，属于动态测试。

确认和验证相关联，但也有明显的区别。Boehm 是这样来描述两者差别的：确认要回答的是，我们正在开发一个正确无误的系统吗？而验证要回答的是，我们正开发的系统是正确无误的吗？

验证测试计划和确认测试计划涉及不同的内容。

(1) 在验证测试计划中要考虑的问题：

- 将进行何种验证活动（需求验证、功能设计验证、详细设计验证还是代码验证）；
- 使用的方法（审查、走查等）；
- 产品中要验证的和不要验证的范围；
- 没有验证的部分所承担的风险；
- 产品需优先验证的范围；
- 与验证相关的资源、进度、设备、工具和责任。

(2) 在确认测试计划中要考虑的问题：

- 测试方法；
- 测试工具；
- 支撑软件（开发和测试共享）；
- 配置管理；

- 风险（预算、资源、进度和培训）。

确认测试和验证测试互相补充，保证最终系统的正确性、完全性和一致性。

### 7.4.3 系统测试过程

若把信息系统开发理解为一个自顶向下、逐步细化的过程，系统测试则是依相反顺序的自底向上、逐步集成的过程。低一级的测试为上一级的测试准备条件。图 7.5 表示了系统测试的 4 个步骤，即单元测试、集成测试、确认测试和综合测试。

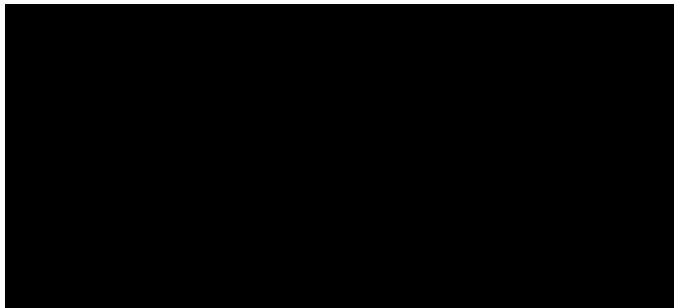


图 7.5 系统测试的步骤

首先对每一个程序模块进行单元测试，以确保每个模块能正常工作。单元测试大多采用白盒测试方法，尽可能发现并消除模块内部在逻辑上和功能上的故障及缺陷。然后，把已测试过的模块组装起来，形成一个完整的软件后进行集成测试，以检测和排除与软件设计相关的程序结构问题，集成测试大多采用黑盒测试方法。确认测试以规格说明规定的需求为尺度，检验开发的软件能否满足所有的功能和性能要求。确认测试完成以后，给出的应该是合格的软件产品，但为了检验开发的软件是否能与系统的其他部分（如硬件、数据库及操作人员）协调工作，还需进行综合测试。

#### 1. 单元测试

单元测试（Unit Testing）又称模块测试，集中检验软件设计的最小单元——模块。在正式测试之前必须先通过编译程序检查，改正所有的语法错误，然后用详细设计描述作指南，对重要的执行通路进行测试，以便发现模块内部的错误。单元测试可以使用白盒测试法，而且可以对多个模块进行同时测试。支持单元测试的工具非常多，如 C++ 语言下的 CppUnit、C++Tester，Java 语言下的 JUnit 等。

一般认为，单元测试和编码属于系统开发的同一个阶段。在编写出源程序代码并通过了编译程序的语法检查以后，通常经过人工测试和计算机测试两种类型的测试。

人工测试源程序可以由编写者本人非正式地进行，也可以由审查小组正式地进行，后者称为代码审查，它是一种非常有效的程序验证技术。审查小组最好由下述四人组成：

- ① 组长，他应该是一个很有能力的程序员，而且没有直接参与这项工程；
- ② 程序的设计者；
- ③ 程序的编写者；
- ④ 程序的测试者。

如果一个人既是程序的设计者又是编写者，或既是编写者又是测试者，则审查小组中应该再增加一个程序员。

审查之前，小组成员应该先研究设计说明书，力求理解这个设计。为了帮助理解，可以先由设计者扼要地介绍他的设计。在审查会上由程序的编写者解释他是怎样用程序代码实现这个设计的，通常是逐个语句地讲述程序的逻辑，小组其他成员仔细倾听他的讲解，并力图发现其中的错误。审查会上进行的另外一项工作，是对照程序设计中常见的错误清单，分析审查这个程序。当发现错误时由组长记录下来，审查会继续进行——审查小组的任务是发现错误而不是改正错误。

代码审查比计算机测试优越的是，一次审查会上可以发现许多错误；用计算机测试的方法发现错误之后，通常需要先改正这个错误才能继续测试，因此错误是一个一个地发现并改正的。也就是说，采用代码审查的方法可以减少系统验证的总工作量。

实践表明，对于查找某些类型的错误来说，人工测试比计算机测试更有效；对于其他类型的错误来说则刚好相反。因此，人工测试和计算机测试是互相补充、相辅相成的，缺少其中任何一种方法都会使查找错误的效率降低。

模块并不是一个独立的程序，因此，必须为每个单元测试开发驱动软件和（或）存根软件。通常，驱动软件也就是一个“主程序”，它接收测试数据，把这些数据传送给被测试的模块，并且输出有关的结果。存根软件代替被测试的模块所调用的模块。因此，存根程序也可以称为“虚拟子程序”。它使用被它代替的模块的接口，可能做最少量的数据操作，输出对入口的检验或操作结果，并且把控制归还给调用它的模块。

## 2. 集成测试

即使各个模块局部工作得很好，也并不能保证它们在一起就能够正常地进行工作：数据可能在通过接口时丢失；一个模块可能对另一个模块产生无法预料的副作用；当模块被连到一起的时候，可能达不到期望的功能；在单个模块中可以接受的不精确性在集成起来以后可能会扩大到无法接受的程度。

集成测试（Integrated Testing）是指在单元测试的基础上，将所有模块按照设计

要求组装成一个完整的系统而进行的测试，也称为联合测试或组装测试。重点测试模块的接口部分，需设计测试过程中所使用的驱动模块或存根模块。集成测试有两种不同的方法：非渐增式测试和渐增式测试。

通常存在进行非渐增式集成测试的倾向，也就是说，使用一步到位的方法来构造程序。所有的模块都预先结合在一起，整个程序作为一个整体来进行测试，得到的结果将可能会混乱不堪，而且会遇到许许多多的问题，错误的修正也非常困难，因为在整个程序的庞大区域中想要分离出一个错误来是很复杂、很困难的。一旦这些错误被修正以后，就可能有新的错误出现，这个过程会继续下去，而且看上去似乎是无止境的。

渐增式集成测试是一步到位方法的对立面。程序先分成小的部分进行构造和测试，这个时候错误比较容易分离和修正，接口也更容易进行彻底的测试，而且也可以使用一种系统化的测试方法。按组装次序渐增式集成测试则有多种方案：自顶向下集成测试、自底向上集成测试和回归测试。

### （1）自顶向下集成测试

自顶向下集成是一种构造程序结构的渐增式实现方法。模块集成的顺序是首先集成主控模块（主程序），然后按照控制层次向下进行集成。隶属于（和间接隶属于）主控模块的模块按照深度优先或者广度优先的方式集成到整个结构中。

如图 7.6 所示，深度优先的集成首先集成在结构中的一个主控路径下的所有模块。主控路径的选择可以是任意的，它与应用程序的特性有关。例如，选择最左边的路径，则模块 M1、M2 和 M5 将会首先进行集成，然后是 M8 或 M6，然后开始构造中间的和右边的控制路径。广度优先的集成首先沿着水平的方向，把每一层中所有直接隶属于上一层模块的模块集成起来，从图中来说，模块 M2、M3 和 M4 首先集成，然后是下一层的 M5、M6 等。

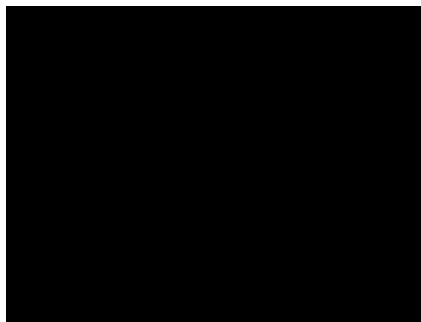


图 7.6 自顶向下集成测试

测试的整个过程由下列五个步骤来完成：

- ① 主控模块作为测试驱动器，所有的存根程序替换为直接隶属于主控模块的模块；
- ② 根据集成的实现方法（如广度优先集成或深度优先集成），下层的存根程序一个一个地被替换为真正的模块；
- ③ 在每一个模块集成时都要进行测试；
- ④ 在完成每一次测试之后，又一个存根程序被真正的模块所代替；
- ⑤ 可以用回归测试来保证没有引出新的错误。

整个过程回到第2步循环继续进行，直至这个系统结构被构造完成时为止。

## （2）自底向上集成测试

自底向上集成测试是从原子模块（如在程序结构的最低层的模块）开始来进行构造和测试的。自底向上的集成测试可以使用以下步骤来实现：

- ① 底层模块组合成能够实现软件特定子功能的簇；
- ② 写一个驱动程序，即一个供测试用的控制程序，来协调测试用例的输入、输出；
- ③ 对簇进行测试；
- ④ 移走驱动程序，沿着程序结构的层次向上对簇进行组合。

这样的集成测试遵循图7.7所示的模式，首先把所有的模块聚集成3个簇：簇1，簇2和簇3。然后对每一个簇使用驱动器（图中的虚线框）进行测试，在簇1和簇2中的模块隶属于Ma，把驱动D1和D2去掉，然后把这两个簇和Ma直接连在一起。类似地，驱动器D3也在模块Mb集成之前去掉。Ma和Mb最后都要和模块Mc一起集成。

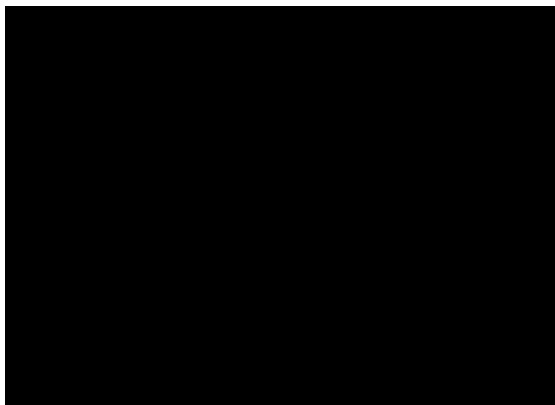


图 7.7 自底向上集成测试

当测试在向上进行的过程中，对单独测试驱动器减少了。事实上，如果程序结构的最上两层是自顶向下集成的，那么所需的驱动数目就会明显地减少，从而对簇的集成会变得简单。



### （3）回归测试

每当一个新的模块被当作集成测试的一部分加进来时，软件就发生了改变。新的数据流路径建立了起来，新的 I/O 操作也可能会出现，而且还有可能激活新的控制逻辑。这些改变可能会使原本工作正常的功能产生错误。在集成测试策略的环境中，回归测试是对某些已经测试过的某些子集再重新进行测试，以保证上述改变不会传播无法预料的副作用。

在更广的环境里，任何种类的成功测试的结果都是发现错误，而错误是要被修改的，每当系统被修改的时候，系统配置的某些方面（程序、文档或者数据）也同时被修改，回归测试就是用来保证由于测试或者其他原因的改动不会带来不可预料的行为或者另外的错误的活动。

回归测试可以通过重新执行所有测试用例的一个子集人工地进行，也可以使用自动化的捕获回放工具来进行。捕获回放工具使得软件工程师能够捕获到测试用例，然后就可以进行回放和比较。回归测试集（要进行的测试的子集）包括三种不同类型的测试用例：

- ① 能够测试软件的所有功能的代表性测试用例；
- ② 专门针对可能会被修改影响的软件功能的附加测试；
- ③ 针对修改过的软件成分的测试。

在集成测试的过程中，回归测试工作可能会变得非常庞大。因此，回归测试集应当设计为只包括涉及在主要的软件功能中出现的一个或多个错误类的测试。如果每当进行一个修改时，就对每一个程序功能都重新执行所有的测试，则不仅不实际而且效率低。

## 3. 确认测试

确认测试（Validation Testing）也称有效性测试或合格性测试（Qualification Testing）。确认测试是在用户参加的基础上，运行软件系统进行测试，以查看系统的功能实现情况及性能上能否满足用户的使用需要。确认测试是软件交付使用前一项很重要的活动，它最终决定用户对该软件的认可程度。确认测试的目的是验证系统的有效性，即验证系统的功能、性能及其他特性是否符合用户需求。系统的功能和性能要求参照需求说明书。

### （1）确认测试内容

确认测试是在开发环境下，由用户参加的测试过程，采用的测试方法为黑盒测试法。

首先，制订测试计划，计划的内容可由开发方起草，最终的定稿要与用户共同协



商，以评定此测试计划是否满足要求。

然后，按照测试计划中的测试步骤，严格审查每一项的测试过程和测试结果，对其进行评定，最后总的测试结果是参照各项的结果确定的。

最后，对系统的相关配置进行审查。包括查看文档是否齐全、内容与实际情况是否一致、产品质量是否符合要求等内容。

## (2) $\alpha$ 测试和 $\beta$ 测试

如果系统是专门为某个用户开发的，则测试工作可以邀请该用户参加，以验证该系统是否满足用户需求。如果系统是为多个用户开发的（如一些公开出售的产品），则要让每个用户都参加确认测试是不切合实际的。因此绝大多数的系统开发者都采用被称为  $\alpha$  测试和  $\beta$  测试的测试方法，尽可能地发现那些看来只有用户才能发现的问题。

$\alpha$  测试是邀请用户参加在开发场地进行的测试，系统环境尽量模拟实际运行环境，由开发组成员或用户实际操作运行。在测试过程中，系统出现的错误或使用中遇到的问题，以及用户提出的修改意见，都由开发者记录下来，作为修改的依据，整个测试过程是在受控环境下进行的。

$\beta$  测试是由部分用户在实际的使用环境下进行的测试。测试过程中开发者不在现场，由用户独立操作，验证程序的各项功能，如界面显示是否友好、交互过程是否方便、功能是否完善、实际使用中还存在什么问题等。同时测试系统性能方面的内容，如系统长时间运行的可靠程度、对出现异常情况的处理能力、兼容性和对环境的适应能力等，从用户使用的角度和真实的运行环境出发，对系统进行测试，用户将发现的问题全部记录下来，反馈给程序开发者，开发者对系统进行必要的修改，并准备最终的发布。

## 4. 综合测试

经过了前面一系列的测试过程，系统的功能已基本符合要求，进行综合测试的目的是为了测试系统安装到实际应用的系统中后，能否与系统的其余部分（如计算机硬件、外部设备、某些支持软件、数据）协调工作，以及对系统运行可能出现的各种情况的处理能力，以确保各组成部分不仅能单独地受到检验，而且在系统各部分协调工作的环境下也能正常工作。在许多文献中，综合测试也称为系统测试，为了与整个信息系统的系统测试区分，将其称为综合测试。

综合测试的任务主要有：测试系统是否能与硬件协调工作，测试与其他系统协调运行的状况。综合测试由若干个不同方面的测试组成，目的是充分运行系统，验证系统是否能正常工作并完成所赋予的任务。综合测试一般包括功能测试、性能测试、安

全测试、恢复测试和安装测试等测试种类。

综合测试是一项比较灵活的工作，对测试人员有较高的要求，既要很了解用户的环境和系统的使用，又要具有各类测试的经验和丰富的软件知识。通过综合测试，保证系统能够在整体的功能和性能上满足用户需求。

5. 测试过程及相应的测试种类

系统测试实际上是由一系列不同的测试组成的。尽管每种测试各有不同的目的，但是所有的测试工作都是为了证实所有的系统元素组装正确，并且执行着为各自分配的功能。下面着重介绍几种测试的种类及它们与各个测试过程中的关系，如表 7.3 所示。

表 7.3 测试过程和测试种类

测试过程 测试种类	开发阶段的测试		产品阶段的测试		
	单元测试	集成测试	$\alpha$ 测试	$\beta$ 测试	综合测试
设计评审		S			
代码审查	M	M			S
功能测试	H	M	M	M	M
结构测试	H	S			
回归测试	S	H			M
可靠性测试		H	M	M	M
强度测试		H			H
性能测试	S	H	M	M	H
恢复测试					M
配置测试		M			M
安全性测试					H
可用性测试		S	M	M	H
安装测试			M	M	M
互连测试	S				M
兼容性测试		M			M
容量测试		H			M
文档测试			S	H	M

说明：M=必要的（Mandatory）；H=积极推荐的（Highly Recommended）；S=建议的（Suggest）。

下面给出表 7.3 中各类测试的说明。

（1）功能测试（Function Testing）：功能测试又称正确性测试，在规定的一段时间内运行软件系统的所有功能，以验证这个软件系统有无严重错误，它检查系统的功能是否符合需求规格说明书。

(2) 可靠性测试 (Reliability Testing): 如果系统需求说明书中有对可靠性的要求, 则需要进行可靠性测试。通常使用平均失效间隔时间 MTBF 与因故障而停机的时间 MTTR 来度量系统的可靠性。

(3) 性能测试 (Performance Testing): 性能测试用来测试在集成系统中各分系统的运行性能, 特别是针对实时系统、嵌入系统。性能测试可以在测试过程的任意阶段进行, 但只有当整个系统的所有成分都集成到一起后, 才能检查一个系统的真正性能。性能测试有时与强度测试相结合, 经常需要软、硬件的配套支持。

(4) 安全性测试 (Security Testing): 安全性测试的目的在于验证安装在系统内的保护机制能否在实际中保护系统并不受非法侵入, 不受各种非法的干扰。系统的安全性测试, 要设置一些测试用例试图突破系统的安全保密措施, 以检验系统是否有安全保密的漏洞。例如, 对密码进行截取和破译, 对系统中的重要文件进行破坏等。测试手段包括各种破坏安全性的方法和工具。

任何系统都很难做到百分之百的安全, 只要能使非法入侵的代价超过被保护信息的价值, 就符合安全性的要求。

(5) 恢复测试 (Recovery Testing): 操作系统、数据库管理系统等都有恢复机制, 即当系统受到某些外部事故的破坏时能够重新恢复正常工作。恢复测试通过各种手段, 强制性地使系统出错, 而不能正常工作, 进而检验系统的恢复能力。如果系统恢复是自动的 (系统本身完成), 则应检验重新初始化、检验点设置位置、数据恢复及重新启动是否正确。如果这一恢复需要人为干预, 则应考虑平均修复时间是否在限定的范围以内。

(6) 文档测试 (Documentation Testing): 文档测试主要检查文档的正确性、完备性和可理解性。这里, 正确性, 是指不要把系统的功能和操作写错, 也不允许文档内容前后矛盾; 完备性是指文档不可以“虎头蛇尾”, 更不许漏掉关键内容; 可理解性是指文档要让大众用户看得懂, 能理解。

(7) 强度测试 (Strength Testing): 强度测试主要检查系统在一些极端条件下的运行情况。因此, 在进行强度测试时需要提供一些超过正常输入量、最大存储能力的测试数据, 查看系统的运行状况。强度测试迫使系统在异常的资源配置下运行, 从而确定系统在功能和性能方面的极限状况。

(8) 可用性测试 (Usability Testing): 可用性测试主要从使用的合理性和方便性等角度对软件系统进行检查, 发现人为因素或使用上的问题。要保证在足够详细的程度下, 用户界面便于使用; 对输入量可容错, 响应时间和响应方式合理可行, 输出信息有意义、正确并前后一致; 出错信息能够引导用户去解决问题; 软件文档全面、正规、确切; 如果产品销往国外, 则要有足够的译本。由于衡量可用性有一定的主观因素, 因此必须以原型化方法等获得的用户反馈作为依据。

(9) 安装测试 (Installation Testing): 安装测试的目的不是找软件错误, 而是找安装错误。在安装软件系统时, 会有多种选择。要分配和装入文件与程序库, 布置适用的硬件配置, 进行程序的联结。而安装测试是要找出在这些安装过程中出现的错误。在一些大型的系统中, 部分工作由软件自动完成, 其他工作则需由各种人员, 包括操作员、数据库管理员、终端用户等, 按一定规程与计算机配合, 靠人工来完成。指定由人工完成的过程也需经过仔细的检查, 这就是所谓的过程测试 (Procedure Testing)。

(10) 互连测试 (Interoperability Testing): 互连测试是要验证两个或多个不同的系统之间的互连性。这类测试对支持标准规格说明, 或承诺支持与其他系统互连的软件系统有效。例如, HP 公司的文件传送存取方法 FTAM、Honeywell 公司 NS/9000 机器上的 FTAM 与 NFT 可以互连。

(11) 兼容性测试 (Compatibility Testing): 这类测试主要想验证软件产品在不同版本之间的兼容性。有两类基本的兼容性测试: 向下兼容和交错兼容。向下兼容测试是测试软件新版本、保留其早期版本的功能的情况; 交错兼容测试是要验证共同存在的两个相关但不同的产品之间的兼容性。

(12) 容量测试 (Volume Testing): 容量测试是要检验系统的能力最高能达到什么程度。在使系统的全部资源达到“满负荷”的情形下, 测试系统的承受能力。

#### 7.4.4 系统测试工具

系统测试需要各种测试工具的支持。测试工具既可以辅助测试工作, 又可以自动化测试过程。在测试过程中应用测试工具可以提高测试效率和质量, 减少测试的开销, 降低测试过程中的重复劳动, 实现测试的自动化。

目前, 测试工具已经有很多, 有针对于系统开发过程中的某个环节的, 有贯穿整个系统开发过程的, 也有成套系列化的。一般而言, 我们将测试工具分为白盒测试工具、黑盒测试工具、测试管理工具几个大类。

##### 1. 白盒测试工具

白盒测试工具一般针对代码进行测试, 对软件的过程性细节做细致的检查, 它允许测试人员利用程序内部的逻辑结构及有关信息设计或选择测试用例, 对程序所有的逻辑路径进行测试。通过在不同点检查程序状态, 确定实际状态是否与预期的状态一致, 测试中发现的缺陷可以定位到代码级。根据测试工具原理的不同, 白盒测试工具又可以分为静态测试工具和动态测试工具。

### (1) 静态测试工具

静态测试工具直接对代码进行分析,不需要运行代码,也不需要代码编译链接、生成可执行文件。静态测试工具评审软件文档或程序,度量程序静态复杂度,检查软件是否符合编程标准,借以发现编写的程序的不足之处,减少错误出现的概率,并根据某种质量模型评价代码的质量,生成系统的调用关系图等。

静态测试工具的代表有 Telelogic 公司的 Logiscope 软件、PR 公司的 PRQA 软件、AutomatedQA 公司的 AQtime 等。

### (2) 动态测试工具

动态测试工具一般采用“插桩”的方式,向代码生成的可执行文件中插入一些监测代码,用来统计程序运行时的数据。与静态测试工具最大的不同,就是动态测试工具要求被测系统实际运行,需要在相对真实的环境下,从多角度观察程序运行时体现的功能、逻辑、行为、结构等行为,以发现其中的错误现象。在动态测试工具中,侧重于对软件功能的测试属于黑盒测试,而对软件内部逻辑结构、测试覆盖率的考虑的则属于白盒测试。

动态测试工具的代表有 Compuware 公司的 DevPartner 软件、Rational 公司的 Purify 系列、AutomatedQA 公司的 AQtime 等。

## 2. 黑盒测试工具

黑盒测试工具适用于黑盒测试的场合,又包括了功能测试工具和性能测试工具。黑盒测试工具的一般原理是利用脚本的录制(Record)/回放(Playback)模拟用户的操作,然后将被测系统的输出记录下来,与预先给定的标准结果比较。黑盒测试工具可以大大减轻黑盒测试的工作量,在迭代开发的过程中,能够很好地进行回归测试。然而,在目前的实际软件测试,特别是软件的功能测试中,黑盒工具的应用还比较困难,对于一些行业性较强的软件项目,很难依靠一个测试工具完成测试工作,但当一一个软件功能比较有规律或者是在回归测试中,那么测试工具往往比较容易实现自动化测试,同时对于复杂的软件项目可以先设计一系列的方案,再提交工具进行自动化测试。

黑盒测试工具的代表有 Mercury Interactive 公司的 WinRunner, Rational 公司的 TeamTest、Robot, Compuware 公司的 QACenter, AutomatedQA 公司的 TestComplete 等。另外,专用于性能测试的工具包括 Mercury Interactive 公司的 LoadRunner、Radview 公司的 WebLoad、Microsoft 公司的 WebStress、Compuware 公司的 QALoad 等, AutomatedQA 公司的 TestComplete 也可以进行性能测试。

### 3. 测试管理工具

测试管理工具是一个可以为企业商业系统提供全面的测试功能的综合测试管理解决方案。它会控制所有的测试工作，以确保一个有组织的、规范文档化的和全面的测试。为了适应数以百计的用户，测试管理工具别具特色之处是它有一个中心数据存储库，在这里所有的用户可以共享并存取主要的信息——测试脚本、缺陷及报告书。测试管理工具把测试计划、测试执行和缺陷跟踪三者有机地结合在一起，同时为了更多的灵活性还采用了开放式测试结构（Open Test Architecture, OTA）。通过测试管理工具，可使测试过程更合理、统一，对软件测试过程实施高效的、标准化的管理。

测试管理工具的代表有 Mercury Interactive 公司的 TestDirect、Rational 公司的 Test Manager、Compuware 公司的 TrackRecord 及 AutomatedQA 公司的 AQ devTeam 等软件。

### 4. 其他测试工具

除了上述的测试工具外，还有一些专用的测试工具，如针对数据库测试的 Test-Bytes、对应用性能进行优化的 EcoScope 等工具。

## 7.5 运行与维护

信息系统交付使用后，便进入运行与维护阶段，此阶段的主要任务是进行系统的日常运行管理，评价系统的运行效率，并根据运行中存在的问题对系统进行维护和处理。其中系统的维护包括对硬件设备的维护和对软件系统的维护，对硬件设备的维护包括对硬件设备进行定期的预防性维护及对突发性的故障进行维修，前者由专职的硬件人员进行，后者则由专职人员或开发方进行。对软件系统的维护是系统维护的重点，通过软件系统的维护，应使系统和数据始终保持最新的正确状态。软件系统的维护类型有四种：正确性维护、适应性维护、完善性维护和预防性维护。下面对其中的系统运行、系统维护、维护过程、维护的特点、可维护性、软件重用与系统维护、信息系统的质量维护等方面的内容加以介绍。

### 7.5.1 系统运行

信息系统通过了综合测试后，就可以着手投入生产运行了。这个过程一般要经过用户培训、系统的转换和系统的运行几个步骤。

## 1. 用户培训

即使信息系统的开发是成功的，但是如果用户不使用，那么成功的信息系统依然无法发挥作用。因此从这个角度看，用户培训也是信息系统开发的一个非常重要的环节。

用户培训的一般内容包括：

- 系统的用途和目标；
- 现有系统与新系统的差别；
- 系统工作概述；
- 如何使用用户手册；
- 与系统有关的信息服务人员和用户人员的义务和责任。

参与培训的人员包括信息系统分析人员和系统用户。按照信息系统的各种文档，系统分析人员可以为最终用户提供用户手册（操作手册和使用指导等，包含系统介绍和详细的操作步骤），并且进行现场培训。开发单位和使用单位必须重视和支持这项工作，必须愿意花时间让各种用户参与到培训中。只有用户使用信息系统，信息系统才能够发挥作用。

用户培训的结果是经过培训的使用者、相应的用户手册及其他文档。只有当最终用户理解和掌握了新系统的操作，才可以开始由旧系统向新系统转换。

## 2. 系统的转换

由于一个组织（企业）的管理工作具有连续进行的特点，所以信息系统也必须连续地进行工作，这就存在一个老的信息系统与新的信息系统的交替过程，即老的信息系统逐渐退出，由新的信息系统来代替。这称为系统的转换（或切换）。

系统转换的方式有直接转换、并行转换和分段转换等不同形式。

### （1）直接转换

直接转换指在某一时刻，老系统停止运行，新系统立即开始运行。这种转换方式简单，最省费用，但风险较大。由于新系统没有使用过，没有真正担负过实际工作，很可能出现事先预想不到的问题。所以这种转换方式不适合重要的或大型的系统，而适用于一些小系统，或者新系统已进行过多次真实测试，或者老系统已完全不能使用等情况。

### （2）并行转换

并行转换指新、老系统并行工作一段时间，并行时间一般为3~6个月，甚至1年。在这段时间里，既可以保持系统工作不间断，又可以对两个系统进行对比，结果



可以互相校对。经过这段时间的考验后，新系统代替老系统。并行转换虽然风险小，但费用高，是一种经常被使用的转换方式。

### （3）分段转换

该转换方式是前两种转换方式的综合。在新系统正式全部运行前，新系统一部分、一部分地代替老系统，这样既避免了直接转换所可能带来的风险，又可避免并行转换将会造成的双倍费用问题。但这种转换方式会增加转换接口，系统各部分之间往往相互联系，当老系统的部分功能转换给新系统去执行时，其余部分仍然由老系统去完成，于是在已转换部分与未转换部分之间就会出现如何衔接的问题。所以分段转换要注意各子系统间、诸功能间的接口问题。一般大型系统采用这种转换方式。

上述3种转换方式如图7.8所示，在实际工作中，往往将上述几种方式互相配合使用。



图 7.8 3 种不同的转换方式

## 3. 系统的运行

系统的运行包括系统的日常操作和维护，应加强对系统的运行管理，使系统正常工作。系统的日常运行管理不仅是机房环境和设施的管理，更主要的是对系统每天的运行状况、数据输入和输出情况，以及系统的安全性、完备性及时、如实地记录和处置。任何一个系统都要经过反复的开发、运行、再开发、再运行的不断完善的过程。

### 7.5.2 系统维护

系统维护是系统生存周期的最后一个阶段，它处于系统投入实际运行以后的时期



中。系统维护需要的工作量很大。虽然在不同应用领域维护成本差别很大，但是平均来说，大型系统的维护成本可以高达开发成本的四倍左右。目前，国外许多系统开发组织把 60% 以上的人力用于维护已有的软件，而且随着系统数量增多和使用寿命延长，这个百分比还在持续上升。将来，维护工作甚至可能会束缚住系统开发组织的手脚，使他们没有余力开发新的系统。

所谓维护，就是在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件的过程。可以通过描述软件交付使用后可能进行的四项活动，具体地定义软件维护。

① 改正性维护。因为软件测试不可能暴露出一个大型软件系统中潜藏的所有错误，所以必然会有第一项维护活动：在任何大型程序的使用期间，用户必然会发现程序错误，并且把他们遇到的问题报告给维护人员。诊断和改正错误的过程称为改正性维护。

② 适应性维护。计算机科学技术领域的各个方面都在迅速发展。一方面，大约每过 36 个月就有新一代的硬件宣告出现，并经常推出新操作系统或旧系统的修改版本，时常增加或修改外部设备和其他系统部件；另一方面，应用软件的使用寿命却很容易超过 10 年，远远长于最初开发这个软件时的运行环境的寿命。因此，适应性维护也就是为了与变化了的环境适当地配合而进行的修改软件的活动，是既必要又经常的维护活动。

③ 完善性维护。当一个软件系统顺利地运行时，常常会出现第三项维护活动：在使用软件的过程中，用户往往提出增加新功能或修改已有功能的建议，还可能提出一般性的改进意见。为了满足这类要求，需要进行完善性维护。这项维护活动通常占软件维护工作的大部分。

④ 预防性维护。为了改进未来的可维护性或可靠性，或为了给未来的改进奠定更好的基础而修改软件时，出现了这项维护活动。这项维护活动通常称为预防性维护，目前这项维护活动相对来说比较少。

从上述关于软件维护的定义不难看出，软件维护绝不仅限于纠正系统在使用中发现的错误，事实上，在全部维护活动中一半以上是完善性维护。国外的统计数字表明：完善性维护占全部维护活动的 50%~66%，改正性维护占 17%~21%，适应性维护占 18%~25%，其他维护活动只占 4% 左右。

应该注意，上述四类维护活动都必须应用于整个软件配置。维护软件文档和维护软件的可执行代码是同样重要的。

### 7.5.3 维护过程

维护过程本质上是修改和压缩了的软件定义和开发的过程。而且，事实上远在提

出一项维护要求之前，与软件维护有关的工作已经开始了。首先，必须建立一个维护组织，随后必须确定报告和评价的过程，而且必须为每个维护要求规定一个标准化的事件序列。此外，还应该建立一个适用于维护活动的记录保管过程并且规定复审标准。

### 1. 维护组织

虽然通常并不需要建立正式的维护组织，然而即使对于一个小的软件开发团体而言，非正式地委托责任也是绝对必要的。每个维护要求都通过维护管理员转交给相应的系统管理员去评价。系统管理员是被指定熟悉一小部分产品程序的技术人员。系统管理员对维护任务做出评价之后，由变化授权人决定应该进行的活动。

在维护活动开始之前就明确维护责任是十分必要的，这样做可以大大减少维护过程中可能出现的混乱。

### 2. 维护报告

应该用标准化的格式表达所有软件的维护要求。软件维护人员通常给用户提供空白的维护要求表，有时称为软件问题报告表，这个表格由要求一项维护活动的用户填写。如果遇到了一个错误，那么必须完整描述导致错误的环境，包括输入数据、全部输出数据及其他有关信息。对于适应性或完善性的维护要求，应该提出一个简短的需求说明书。如前所述，由维护管理员和系统管理员评价用户提交的维护要求表。

维护要求表是一个外部产生的文件，它是计划维护活动的基础。软件组织内部应该制订出一个软件修改报告，它给出下述信息：

- (1) 满足维护要求表中提出的要求所需要的工作量；
- (2) 维护要求的性质；
- (3) 这项要求的优先次序；
- (4) 与修改有关的事后数据。

在拟定进一步的维护计划之前，把软件修改报告提交给变化授权人审查批准。

### 3. 保存维护记录

对于软件生存周期的所有阶段而言，以前的记录保存都是不充分的，而软件维护则根本没有记录保存下来。由于这个原因，所以往往不能估价维护技术的有效性，不能确定一个产品程序的“优良”程度，而且很难确定维护的实际代价是什么。

保存维护记录遇到的第一个问题就是，哪些数据是值得记录的？Swanson 提出了下述内容：

- (1) 程序标识；
- (2) 源语句数；
- (3) 机器指令条数；
- (4) 使用的程序设计语言；
- (5) 程序安装的日期；
- (6) 自从安装以来程序运行的次数；
- (7) 自从安装以来程序失效的次数；
- (8) 程序变动的层次和标识；
- (9) 因程序变动而增加的源语句数；
- (10) 因程序变动而删除的源语句数；
- (11) 每个改动耗费的人时数；
- (12) 程序改动的日期；
- (13) 软件工程师的名字；
- (14) 维护要求表的标识；
- (15) 维护类型；
- (16) 维护开始和完成的日期；
- (17) 累计用于维护的人时数；
- (18) 与完成的维护相联系的纯效益。

应该为每项维护工作都收集上述数据。可以利用这些数据构成一个维护数据库的基础，并且像下面将要介绍的那样对它们进行评价。

#### 4. 评价维护活动

缺乏有效的数据就无法评价维护活动。如果已经开始保存维护记录了，则可以对维护工作做一些定量度量。至少可以从下述 7 个方面度量维护工作：

- (1) 每次程序运行平均失效的次数；
- (2) 用于每一类维护活动的总人时数；
- (3) 平均每个程序、每种语言、每种维护类型所做的程序变动数；
- (4) 维护过程中增加或删除一条源语句平均花费的人时数；
- (5) 维护每种语言平均花费的人时数；
- (6) 一张维护要求表的平均周转时间；
- (7) 不同维护类型所占的百分比。

根据对维护工作定量度量的结果，可以做出关于开发技术、语言选择、维护工作量规划、资源分配及其他许多方面的决定，而且可以利用这些数据分析、评价维护任务。

### 7.5.4 维护的特点

#### 1. 结构化维护与非结构化维护的对比

在图 7.9 中描绘了作为维护要求的结果可能发生的事件流。



图 7.9 结构化维护与非结构化维护的对比

如果软件配置的惟一成分是程序代码，那么维护活动从评价程序代码开始，而且常常由于程序内部文档不足而使评价更困难。诸如软件结构、全程数据结构、系统接口、性能和（或）设计约束等微妙的特点是难以搞清的，而且常常会误解了这一类特点。最终对程序代码所做改动的后果是难于估量的，因为没有测试方面的文档，所以不可能进行回归测试。这种维护方式是没有使用良好定义的方法论开发出来的软件的必然结果。

如果有一个完整的软件配置存在，那么维护工作从评价设计文档开始，确定软件重要的结构特点、性能特点及接口特点；估量要求的改动将带来的影响，并且计划实施途径。然后，首先修改设计且对所做的修改进行仔细复查。接下来编写相应的源程序代码；使用在测试说明书中包含的信息进行回归测试；最后，把修改后的软件再次交付使用。

以上描述的事件构成结构化维护，它是在软件开发的早期应用软件工程方法论的结果。虽然有了软件的完整配置但并不能保证维护中没有问题。但这样做，确实能减少精力的浪费并能提高维护的总体质量。

## 2. 维护的代价

在过去的几十年中，软件维护的费用逐步上升。例如，1970 年用于维护已有软件的费用只占软件总预算的 35%~40%，1980 年上升为 40%~60%，1990 年上升为 70%~80%。

维护费用只不过是软件维护的最明显的代价，其他一些现在还不明显的代价将来可能会更受人们关注。

因为可用的资源必须供维护任务使用，所以耽误甚至丧失了开发的良机，这是软件维护的一个无形的代价。其他无形的代价还有：

- (1) 对一些看来合理的有关改错或修改的要求不能及时满足时将引起用户不满；
- (2) 维护时的改动，在软件中引入了潜伏的故障，从而降低了软件的质量；
- (3) 当必须把软件工程师调去从事维护工作时，将在开发过程中造成混乱。

软件维护的最后一个代价是生产率的大幅度下降，这种情况在维护旧程序时常常出现。例如，据 Gausler 在 1976 年的报道，美国空军的飞行控制软件每条指令的开发成本是 75 美元，然而维护成本大约是每条指令 4 000 美元，也就是说，生产率下降了 50 倍以上。

用于维护工作的劳动可以分成生产性活动（例如，分析评价、修改设计和编写程序代码等）和非生产性活动（例如，理解程序代码的功能、解释数据结构、接口特点和性能限度等）。下列表达式给出了维护工作量的一个模型：

$$M = P + K \times \exp(c - d)$$

其中， $M$  是维护用的总工作量； $P$  是生产性工作； $K$  是经验常数； $c$  是复杂程度（非结构化设计和缺少文档都会增加软件的复杂程度）； $d$  是维护人员对软件的熟悉程度。

上面的模型表明，如果软件的开发途径不好（即没有使用信息系统工程方法论），而且原来的开发人员不能参加维护工作，那么维护工作量和费用将呈指数级增加。

## 3. 维护的问题

与软件维护有关的绝大多数问题，都可归因于软件定义和软件开发的方法有缺点。在软件生存周期的头两个时期没有严格而又科学的管理和规划，几乎必然会导致在最后阶段出现问题。下面列出与软件维护有关的部分问题。

- (1) 理解别人写的程序通常非常困难，而且困难程度随着软件配置成分的减少而

迅速增加。如果仅有程序代码而没有说明文档，则会出现严重的问题。

(2) 需要维护的软件往往没有合格的文档，或者文档资料明显不足。认识到软件必须有文档仅仅是第一步，容易理解的、且与程序代码完全一致的文档才是真正有价值的。

(3) 当要求对软件进行维护时，不能指望由开发人员详细说明软件。由于维护阶段持续的时间很长，因此当需要解释软件时，往往原来写程序的人已经找不到了。

(4) 绝大多数软件在设计时没有考虑将来的修改。除非使用强调模块独立原理的设计方法论，否则修改软件既困难又容易发生差错。

(5) 软件维护不是一项吸引人的工作。这种观念的形成很大程度上是因为维护工作经常遭受挫折。

上述种种问题在现有的未采用信息系统工程思想开发出来的软件中，都或多或少地存在着。虽然不应该把一种科学的方法论看作万应灵药，但是信息系统工程至少部分地解决了与维护有关的问题。

### 7.5.5 可维护性

软件可维护性可以定性地定义为：维护人员理解、改正、改动和改进这个软件的难易程度。提高软件可维护性是支配信息系统工程方法论所有步骤的关键目标。

#### 1. 决定软件可维护性的因素

维护就是在软件交付使用后进行的修改，修改之前必须理解修改的对象，修改之后应该进行必要的测试，以保证所做的修改是正确的。如果是改正性维护，则还必须预先进行调试以确定故障。因此，影响软件可维护性的因素主要有下述三个。

(1) 可理解性。软件的可理解性表现为其他读者理解软件的结构、接口、功能和内部过程的难易程度。模块化、详细的设计文档、结构化设计、源代码内部的文档和良好的高级程序设计语言等，都对改进软件的可理解性有重要贡献。

(2) 可测试性。诊断和测试的难易程度取决于软件容易理解的程度。良好的文档对诊断和测试是至关重要的。此外，软件结构、可用的测试工具和调试工具，以及以前设计的测试用例也都是非常重要的。维护人员应该能够得到在开发阶段用过的测试方案，以便进行回归测试。在设计阶段应该尽力把软件设计成容易测试和容易诊断的。

(3) 可修改性。软件容易修改的程度与采用的设计原理和规则直接相关。耦合、内聚、局部化、控制域与作用域的关系等，都影响软件的可修改性。

上述三个可维护性因素是紧密相关的。维护人员在正确理解一个程序之前根本不

可能修改它；如果不能进行完善的诊断和测试，则表面正确的修改可能引入其他故障。

## 2. 文档

文档是影响软件可维护性的决定因素，由于长期使用的大型软件系统在使用过程中必然会经受多次修改，所以文档比程序代码更重要。

软件系统的文档可以分为用户文档和系统文档两类。用户文档主要描述系统功能和使用方法，并不关心这些功能是怎样实现的；系统文档描述系统设计、实现和测试等各方面的内容。

总的来说，软件系统的文档应该满足下述要求：

(1) 必须描述如何使用这个系统，没有这种描述，即使是最简单的系统也无法使用；

(2) 必须描述如何管理这个系统；

(3) 必须描述系统的需求和设计；

(4) 必须描述系统的实现和测试，以便使系统成为可维护的。

下面分别讨论用户文档和系统文档。

(1) 用户文档。用户文档是用户了解系统的第一步，它应该能使用户获得对系统的准确的初步印象。文档的结构方式应该使用户能够方便地根据需要阅读有关的内容。

用户文档至少应该包括下述五个方面的内容。

① 功能描述：说明系统能做什么。

② 安装文档：说明怎样安装这个系统，以及怎样使系统适应特定的硬件配置。

③ 使用手册：简要说明如何着手使用这个系统。应该通过丰富的例子说明怎样使用常用的系统功能，还应该说明用户操作错误时怎样恢复和重新启动。

④ 参考手册：详尽描述用户可以使用的所有系统设施及它们的使用方法，还应该解释系统可能产生的各种出错信息的含义。对参考手册最主要的要求是完整，因此通常使用形式化的描述技术。

⑤ 操作员指南（如果有系统操作员的话）：说明操作员应该如何处理使用中出现的各种情况。

上述内容可以分别作为独立的文档，也可以作为一个文档的不同分册，具体做法应该由系统规模决定。

(2) 系统文档。所谓系统文档，是指从问题定义、需求说明到验收测试计划这样一系列与系统实现有关的文档。描述系统设计、实现和测试的文档对于理解程序和维护程序来说是非常重要的。与用户文档类似，系统文档的结构也应该能把读者从对系统概貌的了解，引导到对系统每个方面、每个特点的更形式化、更具体的认识中来。



本书前面各章已经较详细地介绍了各个阶段应该产生的文档，此处不再重复。

### 3. 可维护性复审

可维护性是所有软件都应该具备的基本特点，必须在开发阶段保证软件具有前述提到的那些可维护因素。在信息系统工程过程的每一个阶段都应该考虑并努力提高软件的可维护性，在每个阶段结束前的技术审查和管理复审中，应该着重对可维护性进行复审。

在需求分析阶段的复审过程中，应该对将来要改进的部分和可能会修改的部分加以注意并指明；应该讨论软件的可移植性问题，并且考虑可能影响软件维护的系统界面。

在正式的和非正式的设计复审期间，应该从容易修改、模块化和功能独立的目标出发，评价软件的结构和过程；设计中应该对将来可能修改的部分预作准备。

代码复审应该强调编码风格和内部说明文档这两个影响可维护性的因素。

每个测试步骤都可以暗示在软件正式交付使用前，程序中可能需要做预防性维护的部分。在测试结束时进行最正式的可维护性复审，这个复审称为配置复审。配置复审的目的是保证软件配置的所有成分是完整的、一致的和可理解的，而且为了便于修改和管理已经编目归档了。

在完成了每项维护工作之后，都应该对软件维护本身进行仔细认真的复审。

维护应该针对整个软件配置，不应该只修改源程序代码。当对源程序代码的修改没有反映在设计文档或用户手册中时，就会产生严重的后果。

每当对数据、软件结构、模块过程或任何其他有关的软件特点做了改动时，必须立即修改相应的技术文档。不能准确反映软件当前状态的设计文档可能比完全没有文档更坏。在以后的维护工作中很可能因文档不完全符合实际而不能正确理解软件，从而在维护中引入过多的错误。

用户通常根据描述软件特点和使用方法的用户文档来使用、评价软件。如果对软件可执行部分的修改没有及时反映在用户文档中，则必然会使用户因为受挫折而产生不满。

如果在软件再次交付使用之前，对软件配置进行严格的复审，则可大大减少文档的问题。事实上，某些维护要求可能并不需要修改软件设计或源程序代码，只是表明用户文档不清楚或不准确，因此只需要对文档做必要的维护即可。

为了从根本上提高软件的可维护性，人们试图通过直接维护软件规格说明来维护软件，同时，也在大力发展软件重用技术。



#### 4. 软件可靠性与系统可维护性

提高软件可靠性是提高系统可维护性的根本途径。特别是软件可靠性比硬件可靠性更难保证，会严重影响整个系统的可靠性。在许多项目开发过程中，对可靠性没有提出明确的要求，开发商（部门）也不在可靠性方面花更多的精力，往往只注重速度、结果的正确性和用户界面的友好性等，在投入使用后才发现大量的可靠性问题，增加了维护困难和工作量，严重时只有将其束之高阁，无法投入实际使用。

软件可靠性与硬件可靠性之间主要存在以下区别。

（1）硬件有老化损耗现象，硬件失效是物理故障，是器件物理变化的必然结果；软件不发生变化，没有磨损现象，仅有是否陈旧、落后的问题。

（2）硬件可靠性的决定因素是时间，受设计、生产、运用的所有过程影响；软件可靠性的决定因素是与输入数据有关的软件差错，是输入数据和程序内部状态的函数，更多地决定于人。

（3）硬件的纠错维护可通过修复或更换失效的系统重新恢复功能，而软件只有通过重新设计。

（4）对硬件可采用预防性维护技术预防故障，采用断开失效部件的办法诊断故障；而软件则不能采用这些技术。

（5）事先估计可靠性测试和可靠性的逐步增长等技术对软件和硬件有不同的意义。

（6）为提高硬件可靠性可采用冗余技术；而同一软件的冗余不能提高其可靠性。

软件可靠性是关于软件能够满足需求功能的性质，软件不能满足需求是因为软件中的差错引起了软件故障。软件中有哪些可能的差错呢？答案如下。

（1）需求分析定义错误，如用户提出的需求不完整，用户需求的变更未及时消化，软件开发者和用户对需求的理解不同，等等。

（2）设计错误，如处理的结构和算法错误，缺乏对特殊情况和错误处理的考虑等。

（3）编码错误，如语法错误、变量初始化错误等。

（4）测试错误，如数据准备错误、测试用例错误等。

（5）文档错误，如文档不齐全、文档相关内容不一致、文档版本不一致、缺乏完整性等。

所以提高可靠性从原理上看就是要减少错误和提高健壮性。提高软件可靠性的方法和技术有：

（1）建立以可靠性为核心的质量标准；

（2）选择良好的开发方法；

（3）软件重用。

### 7.5.6 软件重用与系统维护

为了从根本上提高软件的可维护性,人们试图通过直接维护软件规格说明来维护软件,同时也在大力发展软件重用技术。软件重用技术是提高系统可维护性的重要方法。

#### 1. 重用的概念

软件重用,是指在两次或多次不同的软件开发过程中重复使用相同或相似软件元素的过程。软件元素包括程序代码、测试用例、设计文档、设计过程、需要分析文档甚至领域知识。对于新的软件开发项目而言,它们或者是构成整个目标软件系统的部件,或者在软件开发过程中发挥某种作用。通常将这些软件元素称为软部件。

为了能够在软件开发过程中重用现有的软部件,必须在此之前不断地进行软部件的积累,并将它们组织成软部件库。这就是说,软件重用不仅要讨论如何检索所需的软部件及如何对它们进行必要的修剪,还要解决如何选取软部件、如何组织软部件库等问题。因此,软件重用方法学通常要求软件开发项目既要考虑重用已有软部件的机制,又要系统地考虑生产可重用软部件的机制。这类项目通常被称为软件重用项目。

按照重要活动是否跨越相似性较少的多个应用领域,软件重用可分为横向重用和纵向重用。横向重用是指重用不同应用领域中的软件元素,如数据结构、分类算法、人机界面组件等。标准函数库是一种典型的、原始的横向重用机制。纵向重用是指在一类具有较多公共性的应用领域之间进行软部件重用。因为在两个截然不同的应用领域之间实施软件重用的潜力不大,所以纵向重用广受瞩目,并成为软件重用技术的主要方面。

不难理解,纵向重用活动的主要关键点是域分析,即根据应用领域的特征及相似性预测软部件的可重用性。一旦根据域确认了软部件的重用价值,即可进行软部件的开发并对具有重用价值的软部件进行一般化,以便它们能够适应新的类似的应用领域。然后,软部件及其文档即可进入软部件库,成为可供后续开发项目使用的可重用资源。这些部件构成了软部件的构造活动。显然,它是一个软部件不断积累、不断完善的渐进过程。随着软部件的不断丰富,软部件库的规模会不断扩大,因此库的组织结构将直接影响软部件的检索效应,特别是当检索手段并不局限于标准函数库所采用的简单名字匹配方法时。可供候选的软部件从库中被检索出来以后,用户还必须理解其功能及行为,以判别它是否真正适用于当前项目。必要时可考虑对某个与期望的功能/行为匹配程度最佳的软部件进行稍许修改,甚至可以将修改后的软部件加进软部件以替代

原有软部件。当然，这要求修改后的软部件比原有软部件具有更高的重用价值。

应用软件系统的规模越做越大越复杂，其可靠性越来越难保证。应用本身对系统运行的可靠性要求越来越高。在一些关键的应用领域，如航空、航天等，其可靠性要求尤为重要，在银行等服务性行业，其软件系统的可靠性也直接关系到自身的声誉和生存发展竞争力。

## 2. 软件重用技术

利用可重用的软件部分来开发软件的技术，称为软件重用技术，它也指开发可重用软件的技术。目前主要有3种软件重用技术。

### (1) 软件组件技术

软件组件技术，即按照一定的规则把可重用的软件成分组合在一起，构成软件系统或新的可重用的软件部分。这种技术的特点是使可重用的软件部分在组合过程中保持不变。它在数学或工程方面的应用软件开发中尤其明显，也在系统软件的输入/输出或存储管理等方面有较成功的应用。使用这种技术需要公用数据库和可重用软件库的支持，前者提供按照公用标准数据模式建立的数据模块，后者提供用于组合的可重用的软件部分。

### (2) 软件生成技术

软件生成技术，即根据形式化的软件功能描述，在已有的可重用的软件部分的基础上，生成功能相似的软件部分或软件系统。使用这种技术需要可重用软件库和知识库的支持，其中知识库用来存储软件生成机理和规则。

### (3) 面向对象的程序设计技术

面向对象的程序设计技术，即传统的面向数据/过程的软件设计方法，把数据和过程作为相互独立的实体，数据用于表达实际问题中的信息，程序用于处理这些数据。程序员在编程时必须考虑所要处理的数据格式，对于不同的数据格式要做同样的处理，或者对于相同的数据格式要做不同的处理，但都必须编写不同的程序。显然，使用传统的软件设计方法，可重用的软件部分比较少。

传统的软件设计方法忽略了数据和程序之间的内在联系。事实上，用计算机解决的问题都是现实世界中的问题，这些问题无非由一些存在并且有一定联系的事物所组成。这些事物称为对象，每个具体的对象都可以用下列两个特征来描述：描述对象所需要使用的数据结构和可以对这些数据进行的有限操作（也就是说，数据结构和对数据的操作）。

### 7.5.7 信息系统的质量维护

信息系统的质量维护包括硬件系统的质量维护和软件系统的质量维护。硬件系统的质量维护主要通过日常设备的运行维护（运行情况记录、故障诊断、排除情况等），定期对系统进行检测、维修，以保证系统具备良好的性能状态。软件系统的质量维护主要是应用系统（软）和数据的质量维护。软件系统的质量维护的情况一般要比硬件系统复杂。

对应用系统的质量维护来说，系统永远不会十全十美，必须不断校正错误或完善提高。如果不进行错误的校正，则用户将失掉对系统的信心；如果不对系统作进一步完善，则用户有可能转向其他的系统。

应用系统的质量维护的方法遵循一个识别、分析、变化和测试的过程。错误的识别及提出改进措施是信息系统质量维护工作的关键。在用户变化需求和维护人员响应这些需求变化之间需要有一定的制度，以保证维护质量符合变化的需要。

数据的质量维护不仅需要输入新的数据来更新数据库，而且还要保证数据的完整性。一个组织机构维护数据质量的能力取决于组织因素和数据因素，这些因素如下所述。

（1）错误影响期。立即产生影响的错误要比长期产生影响的错误更容易引起注意。例如，职工的每月工资错误要比职工的履历错误更容易被发现。

（2）度量工作的规则性。经常定期的数据采集工作比动态的数据收集更易于规则化、制度化，质量更可靠。

（3）用户与提供者的关系。如果提供数据的职能部门与用户没有组织上的联系，则维持高质量的数据就比较困难。

（4）提供者对数据的重视观念。例如，与市场销售人员相比，会计部门有更为严格的数据准确性的观念。

（5）验证的简便性。有些数据可以很方便地被验证是否正确，而有些则比较困难。例如，应收方的账款就很容易与应付方的现金对照检查。

已知这些影响数据的质量因素和维护数据质量的困难之后，就可以建立一些规程来提高维护数据质量的可能性。可以定期进行与实际数据的比较，如与库存数的比较，可以经数据收集的对象来审查他们记录的正确性和完整性，对一些记录还可以由能够鉴别错误的人员定期进行审查等。

## 习 题

1. 什么是系统实施？
2. 系统实施阶段的工作任务有哪些？
3. 如何选择合适的开发平台？
4. 系统转换有哪几种方式？
5. 什么是系统测试？系统测试的主要目的是什么？
6. 白盒测试、黑盒测试方法有何特点？实际应用中应如何选用？
7. 系统维护主要包括哪些方面？
8. 影响软件可维护性的主要因素有哪些？
9. 软件（或系统）的可维护性为什么很重要？
10. 软件重用中所指的软件元素包括哪些？

# 第 8 章 信息系统项目管理

项目管理主要是从建设大型、高费用、进度要求严的复杂系统的需要中逐步发展起来的，用于在项目实施过程中采用科学、合理的管理技术和手段控制项目的进行并保障项目能够按时、保质地完成。

## 8.1 项目管理概述

项目管理是一门关于项目资金、时间、人力、产品等资源控制的管理科学。本节介绍项目的概念、项目管理的范围和特点，以及项目管理知识体系。

### 8.1.1 项目的概念

#### 1. 项目的定义和特征

项目是指在一定的约束条件下（主要是限定资源和时间）具有特定目标的一次任务，如建设项目、科研项目、投资项目等。

项目的主要特征如下所述。

（1）单件性。它决定了达到项目目标的一次性，如建设一项工程、开发一个信息系统等。它不同于其他工业产品的批量性，也不同于其他生产过程的重复性。

（2）具有一定的生命周期。项目的单件性和过程的一次性决定了项目具有一定的生命周期。同时，整个生命周期又明显划分为若干特定阶段。每一阶段都有一定的时间要求和特定的目标，而且都是下一阶段成长的前提，对整个生命周期有决定性的影响。

（3）具有一定的约束条件。项目必须有限定的资源消耗、时间要求和质量规定。

（4）具有特定的目标。

#### 2. 项目管理的含义

项目管理是在特定的组织环境和一定的约束条件下，以最优实现项目目标为目的，按照其内在的逻辑规律对项目进行有效的计划、组织、协调、指挥和控制的系统管理活动。这些活动通常是为了有效管理目标、使各个部门明确工作而制定的一整套的原则、方法、辅助手段和技巧。

项目管理从事项目的规划、监控和跟踪，并为项目组织人员、安排时间、做出预算及保证作业的质量。项目管理有两项宗旨：

（1）参照项目技术和事务两方面的有关文件，对项目管理做出计划并予以说明，以便与实际执行之间进行对照比较；

（2）支持管理技术的进步，以利于更有效地管理人员及其项目。

项目管理的风格是主动性强，沟通能力、协调能力及分析能力是这种风格不可缺少的组成部分。另一关键要素是参照既定目标对实际执行做出评估，此种管理方式的核心乃是对项目目标有清晰的认识和对项目质量有高标准的要求。

## 8.1.2 项目管理的基本内容和特点

### 1. 项目管理的基本内容

项目管理是理顺与项目有关的众多错综复杂问题的一种手段，需要运用项目的专业知识和技术处理好各种技术和人为等因素。项目管理始终贯穿于项目进行过程中的时时刻刻和方方面面，直到项目完成并投入使用。项目管理一般包括以下基本内容。

#### （1）项目定义

项目定义是项目管理过程最初的、也是十分重要的一个阶段。因为此时，项目管理的要求者（项目发起人或项目顾客）与项目经理要就项目的一些重要方面达成一致。为此往往需要回答以下五个重要问题：

- ① 提出的问题或机会是什么？
- ② 项目的目的是什么？
- ③ 为实现这一目的，有哪些目标是必要的？
- ④ 如何确认项目已成功？
- ⑤ 是否存在可能影响项目成功的风险和障碍？

#### （2）项目计划

项目计划不仅告诉我们如何去做工作，而且也是一种制定决策的工具。一个完整的计划会清楚地说明将要做什么、如何去做、由谁来做、何时做、将在什么地方做、将需要什么资源等。项目计划还要包含确定项目完成和成功的标准。

计划可以降低项目实施中的不确定性。事实上，我们并不期待项目像计划的那样精确，计划使我们能考虑到可能的结果并在适当的时候采取必要的纠正措施。

计划可以提高效率。当定义了要做的工作和完成工作所需要的资源后，就可以根据资源安排进度计划，也可以并行安排工作进度而不一定顺序安排。这样就可以最大限度地利用资源，也能比其他方式花更少的时间完成项目工作。

制订计划也会有助于更好地理解项目的目的和目标。即便放弃了已制订的计划，在很多情况下也会从中获益。同时，计划也为实际完成的工作与计划工作之间的对比评价提供了一个参照标准。

### （3）项目执行

项目执行包括几个步骤：确定完成计划规定工作所需的资源（人力、物资和资金等）、组织人员、根据进度计划安排各部门完成的任务和活动的开始与结束时间。

### （4）项目控制

虽然初始进度计划（它规定了将做什么、何时做、谁去做及希望交付出什么）已建立，但是无论项目班子在计划时如何投入，项目工作也不会完全按照计划进行，进度计划有时还会落空。因此在任何情况下，项目经理都必须用一套系统来不间断地监督项目的进展。这个监督系统不仅对项目相对计划的实际完成情况进行汇总，而且对项目的未来加以预测并重新计划，对可能的问题做出预警。问题修正程序和一套正式的变更管理程序是有效项目控制的基础。

### （5）项目结束

结束阶段非常重要，但它经常被管理者忽略，人们总是急于继续下一个项目。在每个项目结束的时候，都有这样几个问题需要回答：

- ① 项目是如它的要求者所要求的那样做的吗？
- ② 项目是按照项目经理要求的那样做的吗？
- ③ 项目班子是根据计划完成项目的吗？
- ④ 获得了哪些有助于今后项目的信息？
- ⑤ 项目管理方法起到怎样的作用，项目班子合作得怎样？

因此，结束阶段需要对所做的工作进行评价，并为今后的项目提供历史信息。

## 2. 项目管理的特点

项目管理既是一门科学又是一门艺术。它之所以被看成一门科学，是因为项目管理是以各种图表、数学计算及其他技术手段为依据的，这些都需要项目管理的硬技术。但是，项目管理也受到政治因素、人际关系因素及组织因素的制约，因而就有项目管理是一门艺术的说法。相互沟通、协商谈判及解决矛盾等即为项目管理艺术所运用的一部分软技术。

项目管理工作应具备三个主要特点：需要达到的技术目标、完成期限及预算。

### （1）技术目标

如果对终极产品或最后提供的服务所要达到的要求不了解，就很难提出计划。在这种情况下，或许能提出某种计划方式，但要制订整个项目计划是不可能的。如果所



定的技术目标只是整个项目的一部分，那么为了有效实施项目管理就需要把项目分割成若干个小项目，从而可将第一个小项目的技术目标作为其终极产品。而且，该终极产品要能经得起客观检验，看其是否具有项目委托人所要求的品质。

### （2）完成期限

完成期限可在项目计划形成前确定，也可在计划被接受后由项目主管与委托人协商而定。不管何种情况，项目的完成应在指定的最后日期以内，任何延误都将带来不良后果。

### （3）预算

预算既可以用需多少资金或多少人员的方式来表示，也可以用需多少资金加多少人员的方式来表示。预算可在项目计划形成前确定，也可根据计划由项目主管与委托人协商而定。

## 8.1.3 项目管理知识体系

美国项目管理学会（Project Management Institute, PMI）的项目管理知识体系（Project Management Body of Knowledge, PMBOK）是对项目管理专业知识的一个总结。PMBOK 把项目管理划分为 9 个知识领域，即项目范围管理、项目时间管理、项目成本管理、项目质量管理、项目人力资源管理、项目沟通管理、项目风险管理、项目采购管理和项目集成管理。近年来，由于 PMBOK 的基本内容跨越行业界限而得到普遍认可，所以受到了国内外专业学术领域的专家学者的广泛重视。对信息系统的建设来讲，它同样具有指导价值，信息系统工程建设的项目团队，包括业主、承包商、工程监理等，也将从中受益。下面简要介绍一下这 9 个知识领域。

### 1. 项目范围管理

该知识领域保证成功地完成项目所要求的全部工作，而且只完成所要求的工作。

- （1）项目启动：对项目或项目的阶段授权。
- （2）范围计划：制定一个书面的范围陈述，作为未来项目决策的基础。
- （3）范围定义：把项目应提交的成果进一步分解成为更小、更易管理的组成部分。
- （4）范围确认：正式地认可项目满足了范围要求。
- （5）范围变更控制：控制项目范围的变更。

### 2. 项目时间管理

该知识领域保证按时完成项目。

- （1）活动定义：识别出为产生项目的提交成果而必须执行的特定活动。

- (2) 活动排序：识别并记录活动之间的相互依赖关系。
- (3) 活动时间估计：估计完成每一个活动将需要的工作时间。
- (4) 制定时间表：分析活动顺序，估计活动时间和资源需求，建立项目进程时间表。
- (5) 时间表控制：控制项目时间表的变更。

### 3. 项目成本管理

该知识领域保证在已批准的预算内完成项目。

- (1) 资源计划：决定执行项目活动所需要的资源的种类（人员、设备、材料）和数量。
- (2) 成本估算：对完成项目活动所需资源的成本进行估计。
- (3) 成本预算：把估算的总成本分配到每一项工作活动中。
- (4) 成本控制：控制项目预算的变更。

### 4. 项目质量管理

该知识领域保证项目的完成能够使需求得到满足。

- (1) 质量计划：找出与项目相关的质量标准，并决定如何满足标准的要求。
- (2) 质量保证：对项目绩效做经常性的评价，使得有信心达到质量标准的要求。
- (3) 质量控制：监视特定的项目结果以判定是否满足相关的质量标准，并找出方法来消除不能满足要求的原因。

### 5. 项目人力资源管理

尽可能有效地使用项目中涉及的人力资源。

- (1) 组织的计划：识别、记录、指派项目的角色、责任和报告关系。
- (2) 人员获得：使项目所需的人力资源得到任命并在项目中开始工作。
- (3) 团队建设：开发个人的和团队的技术来提高项目的绩效。

### 6. 项目沟通管理

该知识领域保证适当、及时地产生、收集、发布、存储和最终处理项目信息。

- (1) 沟通计划：决定项目相关者的信息和沟通的需求，包括谁需要什么信息、什么时间需要及得到信息的方式。
- (2) 信息发布：及时地把所需的信息提供给相关者使用。
- (3) 绩效报告：收集、分发绩效信息，包括状态报告、进度衡量和预测报告。
- (4) 管理上的结束：产生、收集、分发信息，使项目或项目阶段正式地结束。

## 7. 项目风险管理

对项目的风险进行识别、分析和响应的系统化方法，包括使有利事件的机会和结果最大化，以及使不利事件的可能和结果最小化。

- (1) 风险识别：决定哪些风险可能会影响项目并记录风险的特征。
- (2) 风险定性分析：对风险和条件进行定性分析并根据对项目目标的作用排定优先级。
- (3) 风险量化分析：度量风险的可能性和后果，并评估它们对项目目标的影响。
- (4) 风险响应计划：根据影响项目目标的风险制订计划以增加机会和减少威胁。
- (5) 风险监视和控制：监视已知的风险，识别新的风险；执行减低风险计划，在整个项目生命周期中评价它们的有效性。

## 8. 项目采购管理

为达到项目范围的要求，从外部获得物资和服务的过程。

- (1) 采购计划：决定采购的内容和时间。
- (2) 征求货源计划：记录产品需求，识别潜在来源。
- (3) 征求货源：根据需要获得价格、报价、投标、建议书等。
- (4) 来源选择：从潜在的销售商中进行选择。
- (5) 合同管理：管理与销售商的关系。
- (6) 合同结束：合同的完成和结算，包括解决任何遗留问题。

## 9. 项目集成管理

该知识领域保证对项目中的不同的因素能适当协调。

- (1) 制定项目计划：集成、协调全部的项目计划内容，形成一致的、联系紧密的文件。
- (2) 执行项目计划：通过执行其中的活动来执行项目计划。
- (3) 集成的变更控制：在整个项目执行过程中协调变更。

# 8.2 信息系统的项目管理

信息系统工程涉及系统科学、管理科学、计算机科学等方面的知识。信息系统项目管理以传统项目管理的成熟理论和工具为基础，进一步强调管理的系统性、综合性和程序化，以通信为基础，将质量管理、时间进度管理集成到统一的环境中，实现两者的协调，同时保证物质流与信息流的统一。信息系统项目管理更加重视项目的组织方式、人员素质和文档管理。

### 8.2.1 概述

信息系统建设构成一类项目，因此必须采用项目的思想和方法来指导。对于以往信息系统的建设，业界有两个80:20的估计：80%的项目都失败了，只有20%的项目是成功的；在那些失败的项目中，80%的原因是非技术因素导致的，只有20%的项目是由技术因素导致的失败。在这里，非技术因素包括企业业务流程与组织结构的改造问题、企业领导的观念问题、企业员工的素质问题、项目管理问题等。在绝大多数情况下，信息系统项目的失败最终表现为费用超支和进度拖延。不能保证有了项目管理，信息系统的建设就一定能成功，但项目管理不当或根本就没有项目管理意识，则必然导致信息系统建设的失败。

信息系统建设作为一类项目，具有三个鲜明的特点。

(1) 目标不精确、任务边界模糊、质量要求主要由项目团队定义。在信息系统开发中，客户方常常在项目开始时只能提出一些初步的功能要求，没有明确的想法，也提不出确切的需求，因此信息系统项目的任务范围在很大程度上取决于项目组所做的系统规划和需求分析。由于客户方对信息技术的各种性能指标并不熟悉，所以信息系统项目所应达到的质量要求也更多地由项目组定义，客户方则担负着审查任务。为了更好地定义或审查信息系统项目的任务范围和质量要求，客户方可以聘请信息系统项目监理或咨询机构来监督项目的实施情况。

(2) 客户需求随项目进展而变，导致项目进度、费用等不断变更。尽管已经做好了系统规划、可行性研究，签订了较明确的技术合同，然而随着系统分析、系统设计和系统实施的进展，客户的需求不断地被激发，导致程序、界面及相关文档需要经常修改。而且在修改过程中又可能产生新的问题，这些问题很可能经过相当长的时间后才会被发现，这就要求项目经理不断监控和调整项目的计划执行情况。

(3) 信息系统项目是智力密集型和劳动密集型项目。信息系统项目受人力资源的影响最大，项目成员的构成、责任心、能力和稳定性对信息系统项目的质量及是否成功具有决定性的影响。

信息系统项目工作的技术性很强，需要大量高强度的脑力劳动。尽管近年来信息系统辅助开发工具越来越多，但是项目各阶段还需要大量的人工劳动。这些细致、复杂的劳动极容易出错，因而信息系统项目既是智力密集型项目，又是劳动密集型项目。

此外，由于信息系统开发的核心成果——应用软件是不可见的逻辑实体，因此如果其人员发生流动，则对于没有深入掌握软件知识或缺乏信息系统开发实践经验的人来说，很难在短时间里做到无缝承接信息系统的后续开发工作。

另外，信息系统的开发，特别是软件开发渗透了人的因素，带有较强的个人风格。所以为了高质量地完成项目，必须充分发掘项目成员的智力、才能和创造精神，不仅要求他们具有一定的技术水平和工作经验，而且还要求他们具有良好的心理素质和责任心。与其他行业相比，在信息系统开发中，人力资源的作用更为突出，所以必须在人才激励和团队管理问题上给予足够的重视。

由此可见，信息系统项目与其他项目一样，在项目范围管理、项目时间管理、项目成本管理、项目质量管理、项目人力资源管理、项目沟通管理、项目采购管理、项目风险管理和项目集成管理这9个领域都需要加强，特别是要突出人力资源管理的重要性。同时，考虑到软件的特殊性，还需要进行信息系统的配置管理，这主要是文档和版本管理。与其他工程项目一样，研制开发一个信息系统也需要在给定的时间内计划、协调和合理配置与使用各种资源。

对信息系统进行项目管理的重要性有以下四点：

- (1) 可以进行系统的思考，进行切合实际的全局性安排；
- (2) 可为项目人力资源的需求提供确切的依据；
- (3) 通过合理的计划安排对项目进行最优化控制；
- (4) 能够提供准确、一致和标准的文档数据。

## 8.2.2 基本内容与步骤

### 1. 信息系统项目管理的阶段划分

项目管理工作是进展性的，它将生产终极产品的过程与计划，处理变化，控制，采取预防及整改措施等过程融合在一起。信息系统开发的项目管理可分为立项与可行性研究和项目实施管理两个阶段。

#### (1) 立项与可行性研究

信息系统项目开发前期一般分为两步：第一步为初步可行性研究，即进行初步调查，提出项目建议书；第二步为可行性研究，即正式研究阶段。在第一步，项目建议书由项目主管部门批准后，项目就被列入计划，也就是项目立项。接下来就可以开始正式的可行性研究了，项目能否正式实施还有待可行性研究报告是否被审查批准。对一些项目，上述过程可以从简。

可行性研究是在项目开发前期对项目的一种考察和鉴定，它对拟订中的项目进行全面、综合的调查研究，其目的是要判断项目是否可行。信息系统技术可行性研究要从系统开发的计划出发，论述系统开发力量的可行性，同时论证系统方案中所采取的

各种技术手段是否可以实现。信息系统经济可行性研究主要是对项目进行经济评价,分析系统建设投资的可能性,评价系统运行之后给组织带来的效益。信息系统营运可行性研究主要从人力、物力、组织工作等方面论证能否保证项目按计划完成实施,还要说明项目开发后在经济、技术和环境等方面能否使系统正常运行。

## (2) 项目实施管理

信息系统的项目被批准实施之后,就应开始项目实施的管理工作。项目实施管理的目的是通过计划、检查和控制等一系列措施,使系统开发人员能够按项目的目标有计划地进行工作,以便成功地完成项目。项目组的人员组成应面向项目而不是按专业进行组织,一般由项目负责人领导,项目组内可按任务进行再分组。当大型的信息系统项目分为多个子项目进行开发时,需要有一个总的项目管理组负责对各个子项目的公共部分做出指导、协调和管理,各个子项目相应有各自的项目管理小组。项目实施管理的主要内容包括开发管理、测试管理、运行管理和项目后评价管理。

开发管理的主要内容有:制定文档,预计需要的资源,费用估算,安排工作任务和日程,定期做评审,质量保证管理,开发总结报告,处理意外情况等。测试管理的主要内容有:制订测试计划,测试分析并报告,编制用户手册。运行管理的主要内容有:人员的组织与管理,设备和资料管理,财政预算与支出管理,作业时间管理。项目后评价管理的主要内容有:技术水平与先进性评价,经济与社会效益分析,系统的内在质量评价,系统的推广使用价值评价,系统的不足之处与改进意见等。

项目实施管理贯穿于系统分析、系统设计、系统实施、系统维护和评价的整个系统开发过程。

## 2. 信息系统项目实施管理的基本内容

在信息系统项目实施的过程中,其项目管理可按下面5个步骤来进行。

### (1) 任务分解

任务分解是把整个信息系统的开发工作定义为一组任务的集合,这组任务又可以进一步划分成若干个子任务,进而形成具有层次结构的任务群,使任务责任到人、落实到位、运行高效。任务分解是实现项目管理科学化的基础,虽然进行任务分解要花费一定的时间和精力,但是在整个系统开发过程中将会越来越显示出它的优越性。

任务分解包括的内容有:任务设置,资金划分,任务计划时间表,协同过程与保证完成任务的条件。

任务设置是在统一文档格式的基础上详细说明每项任务的内容、应该完成的文档资料、任务的检验标准等;资金划分是根据任务的大小、复杂程度,所需的硬件、软件、技术等多种因素确定完成这项任务所需的资金及分配情况;任务计划时间表是根

据所设置的任务确定完成的时间；协同过程与保证完成任务的条件是指在任务划分时要考虑为了完成该项任务所需要的外部条件和内部条件，即哪些人需要协助、参与该项任务，保证任务按时完成的人员、设备、技术支持、后勤支持具体是什么等。在进行了任务分解之后，将这些任务落实到具体的人，并建立一张任务分解表，在这张表中标明任务编号、任务名称、完成任务的责任人。其中，任务编号是按照任务的层次对任务进行编码，最高级别的任务为1，依次为2，3，…，对任务1的分解为1.1，1.2，1.3，…，对任务2的分解为2.1，2.2，2.3，…，以此类推。

任务分解的主要方法有以下3种。

① 按系统开发项目的结构和功能进行分解，即可以将整个系统开发项目分为硬件系统、系统软件和应用软件系统。硬件系统可分为服务器、工作站、计算机网络环境等，操作中要考虑这些硬件的选型方案、购置计划、购置管理、检验标准、安装调试计划等内容，制定相应的任务；系统软件可划分为网络操作系统软件、后台数据库管理系统、前台开发平台等，操作中要考虑这些软件的选型、配件、购置、安装调试等内容并制定相应的任务；对于应用软件来说可将其划分为输入、显示、查询、打印、处理等功能，包括对系统进行需求分析、总体设计、详细设计、编程、测试、检验标准、质量保证、审查等内容并制定相应的任务。

② 按系统开发阶段进行分解，即按照系统开发中的系统分析、系统设计、系统实施及系统实施中的编程、系统测试、系统安装调试、系统试运行、系统运行等各个阶段分解出每个阶段应该完成的任务、技术要求、软硬件系统的支持、完成的标准、人员的组织及责任、质量保证、检验及审查等内容，同时还可根据完成各阶段任务所需的步骤将这些任务进行更细一级的分解。

③ 将①和②结合起来进行分解。采用这种方法主要是从实际应用出发，兼顾两种方法的不同特点而进行考虑的。

在进行任务分解过程中应特别注意两点。一是分解任务的数量不宜过多，但也不能过少。过多会引起项目管理的复杂性与系统集成的难度；过少会对项目组成员，特别是任务负责人有较高的要求，因而影响整个开发，所以应该注意任务分解的恰当性。二是在任务分解后应该对任务负责人赋予一定的职权，明确责任人的任务、界限及对其他任务的依赖程度，确定约束机制和管理规则。

## （2）计划安排

依据任务分解即可制订出整个开发及项目管理计划，并制订任务时间计划表。开发计划包括以下7种。

① 计算机硬件系统、系统软件配置计划。包括：建立系统基准，配置、选型、购置、安装调试过程，在变化的情况下如何保持系统基准的稳定，建立最终产品的文档。



② 应用软件开发计划。包括：将用户需求转化为相应的项目，软件开发过程；集成软件的过程，测试软件的过程。

③ 测试和评估计划。包括：评估整个系统的集成，整个系统的测试，给用户展示系统的工作情况，评估准备给用户使用的系统。

④ 验收计划。包括：准备验收文档，如何将最终系统提供给用户。

⑤ 质量保证计划。包括：验证开发质量，确定外部产品质量。

⑥ 系统工程管理计划。包括：管理全部系统开发任务，跟踪用户对系统开发的需求。

⑦ 项目管理计划。包括：何时及如何完成任务，建立完成管理计划的策略和标准，协调各种计划。

计划安排还包括培训计划、安装计划、安全性保证计划等。当这些计划制订出来后，可以画出任务时间计划表，表明任务的开始时间、结束时间，表明任务之间的相互依赖程度。任务时间计划表可以按照任务的层次形成多张表，其中系统开发的主任任务可以形成一张表，它是所有子任务时间计划表建立的基础。这些表是所有报告的基础，同时还可以帮助对整个计划实施监控。任务时间计划表的建立可以有多种方法，既可以采用表格的形式，也可以使用图形来表达，还可以使用软件工具，其表达方式取决于实际的应用需求。

### (3) 项目经费管理

项目经费管理是信息系统开发项目管理的关键因素，项目经理可以运用经济杠杆来有效地控制整个开发工作，达到事半功倍的效果。在项目管理中，赋予任务负责人一定职责的同时，还要赋予其相应的经费支配权，并对其进行适当的控制。

在经费管理中要制订两个重要的计划，即经费开支计划和预测计划。

经费开支计划包括：

- ① 完成任务所需的资金分配；
- ② 确认任务的责权，考虑可能的超支情况；
- ③ 确定系统开发时间表及相应的经费开支；
- ④ 如果计划需要变动，则应及早通知项目经理。

预测开支计划包括：

- ① 估计在不同的时间内所需的经费情况；
- ② 了解项目完成所占项目计划的百分比；
- ③ 与经费开支计划相比较；
- ④ 允许项目经理做有计划的经费调整。

### (4) 项目审计与控制

项目审计与控制是整个项目管理的重要部分，它对于整个系统开发能否在预算的范



围内按照任务时间表来完成相应的任务起着关键的作用。相应的管理内容和步骤如下。

① 制定系统开发的工作制度。按照所采用的开发方法，针对每一类开发人员制定出其工作过程中的责任、义务、完成任务的质量标准等。

② 制定审计计划。按照总体目标和工作标准制订出进行审计的计划。

③ 分析审计结果。按计划对每项任务进行审计，分析执行任务计划表和经费的变化情况，确定需要调整、变化的部分。

④ 控制。根据任务时间计划表和审计结果掌握项目的进展情况，及时处理开发过程中出现的问题，及时修正开发工作中出现的偏差，保证系统开发工作的顺利进行。

对于系统开发中出现的变化情况，项目经理要及时与用户和主管部门联系，取得他们的理解和支持，并及时针对变化情况采取相应的对策。

#### （5）项目风险管理

在信息系统开发项目的实施过程中，尽管经过了前期的可行性研究及一系列管理措施的控制，但仍不能过早地确定其效果，因为它与风险联系着，可能达不到预期的效果，费用可能比计划要高，实现时间可能比预期要长，硬件和软件的性能可能比预期要低，等等。因此，任何一个系统开发项目都应具有风险管理，这样才能充分体现出成本分析的优点。在风险管理中应注意的是：

- ① 技术方面必须满足需求，应尽量采用商品化技术，这样可以降低系统开发的风险；
- ② 开销应尽量控制在预算范围之内；
- ③ 开发进度应尽量控制在计划之内；
- ④ 应尽量与用户沟通，不要做用户不知道的事情；
- ⑤ 充分估计到可能出现的风险，注意倾听其他开发人员的意见；
- ⑥ 及时采纳减少风险的建议。

总之，风险管理也是项目管理的重要内容，是项目经理的特别职责。

风险管理过程可以划分为以下4个步骤。

① 第一步，风险辨识。首先列出一个潜在问题表，然后再考虑其中有哪些问题会出现风险。风险的确定应听取技术专家和广大用户的意见。潜在的风险源包括：

- 在总体规划和系统分析阶段所进行的需求分析不完全、不清楚、不稳定、不可行，最终影响软件集成和系统集成；设计结果的可用性、可实施性、可测试性较差，影响系统的后续开发工作；
- 在程序设计过程中，可能出现的各部分之间的不一致性或系统的可扩展性较差；
- 在整个开发过程中，遇到困难和问题时，开发人员可能出的矛盾和不协调性将影响系统开发的质量和开发进度；

- 在实施项目管理过程中，计划的准确性、可监控性、经费运用及分配情况等都将对整个开发工作产生影响。

② 第二步，风险分析。对辨识出的风险进行进一步的确认后分析风险概况，即假设某一风险出现后，分析是否会有其他风险出现；或者假设这一风险不出现，分析它将会产生什么情况，然后确定主要风险出现最坏情况后，如何将此风险的影响降低到最小，同时确定主要风险出现的数量及时间。

③ 第三步，风险缓和。通过对风险的分析确定出风险的等级，对高级的风险要制定出相应的对策，采取特殊的措施予以处理，并指定专人负责重要风险项目的实施，同时在风险管理计划中进行专门的说明。

④ 第四步，风险跟踪。对辨识后的风险在系统设计开发过程中进行跟踪管理，确定还会有哪些变化，以便及时修正计划。其具体内容包括：

- 实施对重要风险的跟踪；
- 每月对风险进行一次跟踪；
- 风险跟踪应与项目管理中的整体跟踪管理相一致；
- 风险的内容和对项目开发的影响应随着时间的不同而相应地变化。

通常，影响项目内在风险的因素有 3 种：项目的规模、业务的结构化程度和项目的技术难度。把这 3 种因素的高低（或大小）组合起来，将产生 8 种可能的项目风险估计，如表 8.1 所示。

表 8.1 项目风险估计表

技术难度	结构化程度	规 模	风 险
低	高	大	低
		小	很低
	低	大	中等
		小	中、低
高	高	大	中等
		小	中低
	低	大	很高
		小	高

信息系统项目管理中的风险管理是根据项目风险水平进行组织和管理的，其管理可采用以下措施和技术。

① 项目组与用户结合的外部结合措施和技术，如建立用户项目管理组织、用户参加的项目小组和用户指导委员会。

② 项目组参与协调工作的内部结合措施和技术，如建立项目评审会、备忘录并使项目组参与决策。

③ 任务结构化、条理化的规范的计划措施和技术，如关键路线图、抓重大事件及项目审批程序等。

④ 估计项目进程的规范化控制措施和技术，如建立具有差异分析的一系列正式的状态报告等。

通常，任务的结构化程度越低，越需要外部与用户的高度结合。难度大的高技术开发项目，往往采用高度的内部项目结合措施和规范化很低的计划和控制。项目风险的管理对策如表 8.2 所示。

表 8.2 项目风险的管理对策表

项目特征	业务的结构化程度	高				低			
	采用的技术难度	高		低		高		低	
	项目规模	大	小	大	小	大	小	大	小
处理风险的项目管理技术	规范化的计划和控制	大量使用	大量使用	大量使用	中高度使用	低中程度使用	少量使用	少量使用	少量使用
	外部结合	低				高			
	内部结合	高		低		高		低	

对信息系统的建设来说，风险管理十分重要，因其涉及各方的开发人员和广大的最终用户的利益。为了保证系统开发的顺利进行，除了要建立一整套的管理职责和规范，坚持将一种正确的开发方法贯穿始终外，还要做好各类人员的思想沟通，使开发项目组的全体人员自始至终都能保持一个声音说话。

8.3 信息系统项目时间管理

“按时、保质地完成项目”大概是每一位项目经理最希望做到的，但工期拖延的情况却时常发生。因而合理地安排项目时间是信息系统项目管理中一项关键的内容，其目的是保证按时完成项目，合理分配资源，发挥最佳的工作效率。时间管理的主要工作包括定义项目活动、活动排序、每项活动的合理工期估算、安排项目进度表、项目进度控制等内容。

8.3.1 时间管理流程

1. 项目活动定义

项目定义阶段必须明确开展的特定工作，以便生产各种项目交付产品。项目工作

被分解为更小、更易管理的工作包（也叫活动或任务），这些工作包应该是能够保障完成交付产品的可实施的详细任务。在信息系统项目实施中，要将所有活动列成一个明确的活动清单，并且让项目团队的每一个成员都能够清楚有多少工作需要处理。活动清单应该采取文档的形式，以便于项目其他过程的使用和管理。当然，随着项目活动分解的深入和细化，工作分解结构可能会需要修改，这也会影响项目的其他部分，例如，成本估算，在更详尽地考虑了具体活动后，成本可能会有所增加。因此在完成活动定义后，要更新项目工作分解结构上的内容。

## 2. 活动排序

在产品描述并列成活动清单的基础上，要找出信息系统项目活动之间的依赖关系和特殊领域的依赖关系、工作顺序。在这里，既要考虑团队内部希望的特殊顺序和优先逻辑关系，也要考虑内部与外部、外部与外部的各种依赖关系及为完成项目所要做的一些相关工作，例如，在最终的硬件环境中进行软件测试等工作。

设立项目里程碑是排序工作中很重要的一部分。里程碑是项目中关键的事件及关键的项目子目标的完成时间，是项目成功的重要因素。里程碑事件是确保完成项目需求的活动序列中不可或缺的一部分。例如，在开发项目中，可以将需求的最终确认、产品移交等关键任务作为项目的里程碑。

在进行项目活动关系的定义时一般采用优先图示法、箭线图示法、条件图示法和网络模板图示法这4种方法，最终形成一套项目网络图。其中比较常用的方法是优先图示法，也称为单代号网络图法。

## 3. 活动工期估算

信息系统项目工期估算是根据项目范围、资源状况计划列出项目活动所需要的工期。估算的工期应该现实、有效并能保证质量。所以在估算工期时要充分考虑活动清单所列的内容、合理的资源需求、人员的能力因素及环境因素对项目工期的影响。在对每项活动的工期估算中应充分考虑风险因素对工期的影响。项目工期估算完成后，可以得到量化的工期估算数据，然后将其文档化，同时完善并再次更新活动清单。

一般来说，工期估算可采取以下4种方式。

（1）专家评审形式。由有经验、有能力的人员进行分析和评估。

（2）模拟估算。使用以前类似的活动作为参照系和未来活动工期的估算基础，计算评估工期。

（3）定量型的基础工期。当产品可以用定量标准计算工期时，则采用计量单位为基础数据进行整体估算。

(4) 预留时间。工期估算中预留一定的比例时间作为冗余时间，以应付项目风险。随着项目进展，冗余时间可以逐步减少。

#### 4. 安排项目进度表

项目的进度计划意味着明确定义信息系统项目活动的开始和结束日期，这是一个反复确认的过程。项目进度表的确定应根据项目网络图、估算的活动工期表、资源需求与共享情况、项目执行的工作日历、进度限制、最早和最晚时间、风险管理计划、活动特征等统一考虑。

进度限制即指根据活动排序考虑如何定义活动之间的进度关系。进度限制一般有两种形式：一种是加强日期形式，以活动之间的前后关系限制活动的进度，如一项活动不早于某活动的开始或不晚于某活动的结束；另一种是关键事件或主要里程碑形式，以定义为里程碑的事件作为要求的时间进度的决定性因素，制订相应的时间计划。

在制订项目进度表时，先以数学分析的方法计算每个活动最早开始和结束时间，以及最迟开始和结束时间，据此得出时间进度网络图，再通过资源因素、活动时间和可冗余因素调整活动时间，最终形成最佳项目进度表。

关键路径法（Critical Path Method, CPM）是时间管理中很实用的一种方法，其工作原理是：为每个最小任务单位计算工期，定义最早开始和结束日期，以及最迟开始和结束日期，按照活动的关系形成顺序的网络逻辑图，找出必需的最长路径即为关键路径。在此基础上可进行时间压缩，时间压缩是指针对关键路径进行优化，结合成本因素、资源因素、工作时间因素、活动的可行进度因素对整个计划进行调整，直到关键路径所用的时间不能再压缩为止，最后得到最佳时间进度计划。

#### 5. 进度控制

进度控制的作用主要是监督进度的执行状况，及时发现和纠正偏差、错误。在进度控制中，要考虑影响项目进度变化的因素、项目进度变更对其他部分的影响因素、进度表变更时应采取的实际措施等。

### 8.3.2 工程进度管理工具和技术

项目时间管理中最主要的内容是进度管理与控制，工程进度管理同时也是信息系统工程中最困难的任务。不同的工程或项目使用不同的软、硬件平台及开发方法，这使任务的估算与调度也变得复杂起来。工程进度管理包括任务分解和任务评定。这当中的许多任务是可以并行操作的，调度者必须配备和协调这些并行任务并进行组织，

从而最优地使用人力和物力。必须避免由于某个关键的任务未完成而导致整个工程延期的情况。在估算工程进度时,管理者不应把每个阶段或步骤看成是独立的且与问题无关的。因为在整个项目过程中工作的某些人员也许会生病或离职,硬件可能会出现故障,基本支撑软件或硬件可能延迟交付。如果工程又新、技术又先进,则其中某些部分可能会出现更大的困难而使工程超过原定的工期。估算的基本经验是估算时先认为没有什么问题发生,而进行正常估算,然后为解决预料中的问题再估算,最后另加一个意外因子以解决未预料到的问题。意外因子取决于工程的类型,所面临的问题(如交付日期、标准)及参加项目的人员的经验、素质等。

有一个大致的原则:在正常情况下,系统分析与系统设计的工作时间分别都是编程工作时间的两倍。为了估算整个工程所需的全部时间,就要优先估算系统的大小,然后用所期望的程序员生产率去除,从而得到完成该工程所需的程序员人月数。工程进度管理的输出结果常常是一组包括工作分解、任务依赖及人员分配的活动网络图和进度图。

### 1. 活动网络图

考虑表 8.3 所示的一组活动。表中  $T_i$  ( $1 \leq i \leq 12$ ) 是第  $i$  个子任务。对应于该子任务的还有其工期及依赖的前期子任务。例如,  $T_3$  依赖于  $T_1$ , 换句话说, 仅当  $T_1$  完成后,  $T_3$  才能开始执行。当然, 实际估算工期时, 还应考虑某些意外因素以弥补那些不可预测的延迟, 但对工程管理来说, 估算出错并出现不希望的延期是一个生活常识。所以常常应该允许估算出现某些偏差。

表 8.3 任务工期及其依赖

任务	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$	$T_8$	$T_9$	$T_{10}$	$T_{11}$	$T_{12}$
工期(天)	8	15	15	10	10	5	20	25	15	15	7	10
依赖于			$T_1$		$T_2 T_4$	$T_1 T_2$	$T_1$	$T_4$	$T_3 T_6$	$T_5 T_7$	$T_9$	$T_{11}$

用图 8.1 所示的活动网络图来表示表 8.3 所示的任务依赖关系。图中的矩形节点表示任务, 其上边标明该任务的工期。八边形节点表示活动的路标或工程的里程碑, 其上边也标明前期依赖活动至此所希望的完成日期, 要从一个里程碑向下一个里程碑前进, 就必须完成所有通向该里程碑的前期任务。例如, 在图 8.1 中, 只有当  $T_3$ 、 $T_6$  都完成, 到达里程碑  $M_4$  之后,  $T_9$  才能开始执行。整个工程的工期可通过考察活动网络图中的最长路径或关键路径来估算。图 8.1 中的最长路径就是用阴影表示的那条路径, 即  $T_1 \rightarrow M_1 \rightarrow T_3 \rightarrow M_4 \rightarrow T_9 \rightarrow M_6 \rightarrow T_{11} \rightarrow M_8 \rightarrow T_{12}$  的路径。关键路径上所有的进度都是后者依赖前者, 这条路径上的每个关键活动工期的拖延都会导致整个工程工期的延迟。然而, 非关键路径上的任务的拖延却并不引起全局调度的工期延迟。例如, 图中

T<sub>8</sub> 的拖延就不大可能影响整个工程的工期。

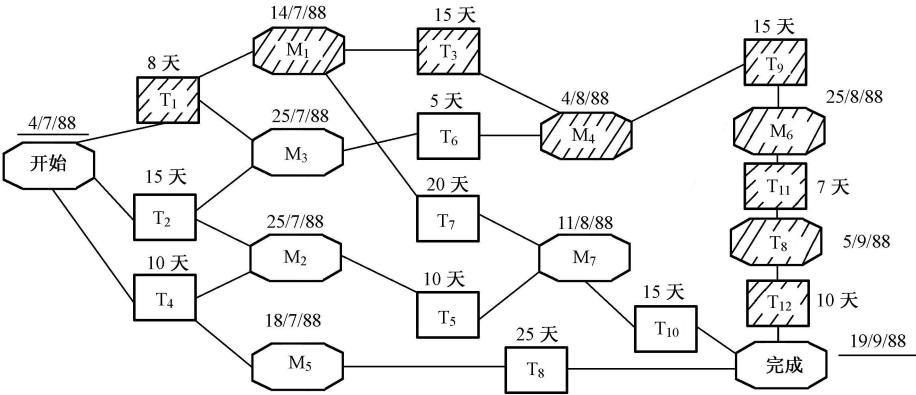


图 8.1 活动网络图

计划评价与审查技术（Program Evaluation and Review Technique，PERT）图是工程活动中更令人熟悉的一种表示形式，它并不对每个任务都进行近似悲观的估算，而是寻找最优估算。因此，它可能有许多潜在的关键路径，这取决于对这些任务的估算排列；而且，寻找关键路径的分析是很复杂的，一般要用计算机来自动求解。

活动网络图不仅可用来进行工期的估算，还可用来分配工程的工作。管理者可从图上直观、明显地看到任务之间的依赖关系。有时，还可修改系统设计以缩短关键路径。通过削减等待活动完成的空耗时间，便可缩短工期。另外，也可用 PC 来自动生成活动网络图。若给定一个初始的开始日期及每个子任务的工期，则所有任务的完成日期便可自动计算出来，对于计划的其他表示再稍加努力也可通过 PC 或其他方式计算出来。

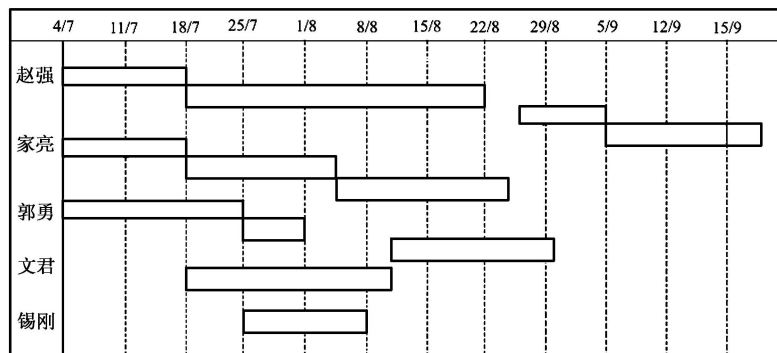
2. 进度图

在项目时间管理中，管理者必须考虑资源分配，这里主要是指把具体任务分配给具体的工作人员，如程序员、系统分析员等。人员的任务分配可使用图 8.2（a）所示的表格，而人员的工作进度计划可使用图 8.2（b）所示的进度图。它清楚地表明了每个人做的工作及其时间进度。并非所有参加项目的人员都要占用整个工程的所有时间，因为在整个工程期间他们可能度假、参加其他工程的培训或从事一些其他活动。大型组织常常雇用许多专家来承担工程的工作，这可能引起调度的问题。如果专家承担工作的工程延期，就可能产生连锁反应，使其他工程因该专家不能参加而导致工程的延期。



任务	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	……	T <sub>12</sub>
程序员	家亮	郭勇	家亮	赵强	锡刚	郭勇	文君	……	赵强

(a) 人员的任务分配图



(b) 人员的工作进度计划图

图 8.2 任务的分配与进度

一般工程的初始调度不可避免地会有错误，这时就应把估算与已经过去的实际时间进行比较，并作为修正后续工程调度的基础。为了缩短关键路径的长度，当知道实际情形以后，修正活动图并重新划分工程的后续任务也是很重要的。

## 8.4 信息系统项目人力资源管理

信息系统建设是智力密集型、劳动密集型项目，因而受人力资源影响最大，项目成员的结构、责任心、能力和稳定性对项目的质量以及是否成功有着决定性的影响。人在信息系统项目中既是成本，又是资本。一般来说，人力成本占信息系统项目成本构成的主要部分，这就要求必须从成本角度去衡量人力资源，尽量使人力资源的投入最小、产出最大。

### 8.4.1 项目管理的组织机构

要保证信息系统开发工作的顺利启动，首先要建立项目的组织机构——项目组。项目组可以由负责项目管理和开发的不同方面的人员组成，由项目组长或项目经理来领导。一般来说，可以根据项目经费的多少和系统的大小来确定相应的项目组。项目组根据工作需要可设若干小组，小组的数目和每个小组的任务可以根据项目规模、复杂程度和周期长短来确定，可以设立的小组有过程管理小组、项目支持小组、质量保证小组、系统工程小组、开发与测试小组、系统集成与安装调试小组等。一个好的项



项目组不一定能保证项目的成功，但一个差的项目组将肯定会导致项目的失败。因此，在建立项目组时要充分利用项目组中每个成员的特长，坚持将正确的开发方法贯穿项目始终。

### 1. 项目经理

项目经理是整个项目的领导者，其任务是保证整个开发项目的顺利进行，负责协调开发人员之间、各级最终用户之间、开发人员和广大用户之间的关系。项目经理拥有资金的支配权，可以把资金作为强有力的工具来进行项目管理，对其资金运用情况可采用定期向上级汇报等方法进行合理监督。

项目经理在实施项目领导工作时，要时刻注意所开发的系统是否符合最初制定的目标，在开发工作中是否运用了预先选择的正确的开发方法，哪些人适合于做哪些工作等。只有目的明确、技术手段合适、用人得当，才能保证系统开发的顺利进行。

对于小型项目，项目经理可以独立进行工作，直接管理各类开发技术人员，必要时可以求得外部机构的支持；对于中型项目，应划分出各个任务的界限，由不同的人去管理，项目经理通过这些人来实施各项管理工作；对于大型项目，应有专门的管理机构进行辅助管理，项目经理应能保证其思想的正确实施，并通过管理机构对开发技术人员的工作实施管理，同时注意对其产品的审核。

### 2. 过程管理小组

过程管理小组负责整个项目的成本及进度控制、配置管理、安装调试、技术报告发布与培训支持等任务。过程管理小组是一个综合性的机构，其作用是保证整个开发项目的顺利进行。

### 3. 项目支持小组

项目支持小组的任务是保障后勤支持，它要及时提供系统开发所需要的设备、材料，负责进行项目开发的成本核算，负责合同管理、安全保证等工作。对大型项目来说，由于涉及的资金巨大、开发人员众多、材料消耗也多，所以项目支持小组尤其要进行科学的管理。

### 4. 质量保证小组

质量保证小组的任务是及时发现影响系统开发质量的问题并给予解决。问题发现得越早，对整个项目的影响越小，项目成功的把握就越大。

## 5. 系统工程小组

由于信息系统开发是一项系统工程，因此可以按照工程的一般特性，用系统的观点制订出系统开发各个阶段的任务计划，这是系统工程小组的工作职责，即将整个开发过程按阶段划分出若干个任务，规定好每个任务的负责人、任务的目标、检验标准、完成任务的时间等。只有明确每一项任务的责、权、利，才能使开发工作进行顺利。

## 6. 开发与测试小组

开发与测试小组的任务是充分利用系统开发的一些关键技术、开发模型及一些成熟的商品软件从事各子系统的开发与集成，并对各子系统进行测试。这是整个开发项目的关键，因此要组织好该小组的成员，并采用统一的方法和标准进行工作。

## 7. 系统集成与安装调试小组

系统集成是对整个信息系统进行综合的过程，该小组成员要在充分注意软件、硬件产品与所开发的信息系统之间的结合，注意最大限度地保证系统可靠性及发挥系统的最高效率的前提下完成信息系统的软件、硬件等各方面的集成，并做好整个系统的测试与安装调试工作。

### 8.4.2 项目角色及其职责

项目开发团队的组织是信息系统成功开发的重要因素之一。信息系统的建设是比较大的工程项目，必须进行任务的分解，并由不同的人员共同来完成。项目开发团队的组建一般包括项目经理（项目负责人）、系统分析员、系统设计员、数据库系统管理员、系统管理员、程序设计员、文档管理员等。另外，信息系统项目的团队还要邀请部分投资方的业务人员参加。项目开发团队中的各种角色成员在项目开发的过程中分担着不同的工作。他们需要相互协作，共同来完成系统的开发工作。

#### 1. 项目负责人

项目负责人负责管理项目的开发活动和开发方向。他应该具有很强的管理才能、丰富的组织经验和协调能力，熟悉项目开发过程中的转折点，在参与项目的各方之间找到一个让各方都满意的方案。项目负责人负责下述工作：

（1）制订项目计划，明确各项具体任务需要的时间，控制项目的进度；

（2）确定开发所用的技术和方法，并在项目的进行过程中组织应用这些技术完成具体的工作；

(3) 有计划地分配现有的各种资源, 合理安排技术人员的工作, 正确处理各种资源的短缺和技术人员离开项目团队的情况;

(4) 掌握项目参与各方的实际需求, 协调项目参与各方的关系;

(5) 控制项目的规模, 随着信息系统项目的开发, 会出现功能需求和系统规模不断增长的情况, 项目负责人需要合理地控制项目的规模;

(6) 正确地评价团队中的每一位成员, 正确地评价他们的工作成绩, 并给予适当的激励, 肯定团队中每一位成员的贡献。

## 2. 系统分析员

系统分析员负责确定具体的业务需求, 并正确地传达给系统设计员和其他开发人员。系统分析员应该具备丰富的相关领域业务知识, 能够与企业的业务负责人员很好地交流, 并明确地表达实际的业务需求。系统分析员的工作包括下列内容:

- (1) 设计业务需求调查问卷;
- (2) 与业务人员进行交流, 明确具体的业务需求;
- (3) 了解企业组织结构及人员配备;
- (4) 明确企业内部职能的划分及与其他部门的关系;
- (5) 获取相关业务的原始单据和报表;
- (6) 确定需要输入和输出的内容及数据的处理流程;
- (7) 明确数据间的计算关系;
- (8) 参与系统使用人员的培训。

## 3. 系统设计员

系统设计员是信息系统项目团队中非常重要的角色, 主要负责信息系统开发的总体设计和详细设计。系统设计员不仅要具备相关领域业务知识, 理解具体的业务需求, 而且要具备丰富的计算机硬件和软件知识, 设计如何实现系统分析中提出的业务需求。系统设计员要完成下列工作:

- (1) 根据业务需求, 设计目标系统的运行模式及业务流程;
- (2) 评估并选择系统的网络设备、硬件设备和相关软件;
- (3) 确定目标系统的功能结构;
- (4) 完成数据库数据模型的设计;
- (5) 确定数据编码方案;
- (6) 对系统功能结构中的模块进行处理和输入/输出设计。

#### 4. 数据库系统管理员

数据库是信息系统的重要组成部分。数据库系统管理员负责数据库系统的正常使用管理，保证数据库系统的安全性和保密性。数据库系统管理员应该非常熟悉所应用的数据库系统，其负责的主要工作如下：

- (1) 数据库系统的逻辑设计和物理实现；
- (2) 数据库系统的升级；
- (3) 采用适当的措施对数据库系统进行加密，保证只有经过授权的用户才能够使用相应的数据；
- (4) 确定衡量数据库系统性能的指标，并监控数据库系统的性能及规模增长，保证数据库系统的正常运行；
- (5) 确保数据库系统正确的备份和恢复；
- (6) 数据库系统的日常管理。

#### 5. 系统管理员

系统管理员也是信息系统项目管理团队中很重要的角色，主要负责计算机系统的管理，保证计算机系统的安全。系统管理员必须具有丰富的计算机硬件和软件知识，并能够随时投入工作。系统管理员的工作包括如下内容：

- (1) 硬件系统的安装和软件系统的配置；
- (2) 硬件、软件系统的升级；
- (3) 创建系统安全机制，保证系统的安全运行；
- (4) 确定衡量系统性能的指标，并监控系统的性能；
- (5) 计算机系统的日常管理和突发问题的解决。

#### 6. 程序设计员

程序设计员的工作是进行程序设计，即使用应用开发工具来实现系统设计中的内容。程序设计员应该熟悉系统的硬件环境，熟练掌握所使用的数据库系统和计算机程序设计语言。程序设计员的工作包括：

- (1) 按照统一的规范书写程序源代码；
- (2) 进行系统交付使用前的程序调试；
- (3) 负责合同所规定的系统维护期内的程序维护。

## 7. 文档管理员

在信息系统的开发过程中，存在着普遍不愿意在开发阶段书写文档的不良现象。但实际情况表明，没有完整系统的文档会给未来系统的维护带来巨大的困难，也是信息系统项目管理的一种失败。配备专门的文档管理员来负责项目文档的书写和管理是一种比较好的选择。文档管理员应该具有比较强的写作能力，且具有极大的耐心和细心。文档管理员主要负责如下工作：

- (1) 参照统一的文档书写规范，撰写及整理项目开发各阶段的文档；
- (2) 对文档进行分类，并编制文档目录；
- (3) 负责文档的日常管理。

## 8. 业务人员

信息系统的开发需要系统开发人员和系统使用人员之间的相互配合。开发人员和系统使用人员的配合与协作非常重要，这主要源于以下两个方面的原因：一方面是信息系统的开发人员往往对计算机系统非常熟悉，但是对具体业务不是很了解，所以一般从计算机技术的角度考虑问题，在进行系统的分析与设计时不容易正确理解系统的需求；另一方面，系统的使用人员对具体业务非常熟悉，但是对信息系统的开发方法不是很了解，可能会提出计算机系统难以实现的要求。系统的开发人员和系统使用人员必须相互配合，反复商讨，只有这样才能做好信息系统的分析与设计工作。在项目团队中，业务人员主要负责下述工作：

- (1) 协助系统分析员了解企业的组织机构、人员配备、组织内部的职能划分及各部门之间的关系；
- (2) 直接或协助系统分析员填写相关需求调查表；
- (3) 提供相关的原始单据和报表；
- (4) 提供相关的数据指标体系及相应的计算关系公式；
- (5) 协调应用方与项目组及其他各方之间的关系。

在信息系统的开发过程中，上述各角色是必须的，但工作的划分不是绝对的。例如，在很多应用系统中会出现这样的情况：系统管理员与数据库系统管理员由同一人担任，不一定配备专门的文档管理员，系统设计员同时负责系统的分析等。另外，在有些关键的技术问题上，还可能外聘相关领域的专家，请他们提供帮助和提出建设性的意见。

### 8.4.3 管理中的协调工作

在信息系统开发的项目管理中，存在着大量的协调工作，主要涉及需求方与开发方的关系、需求方项目管理人员与使用人员及决策层的关系、项目管理人员与软件开发人员的关系、系统性能与灵活应变的关系四个方面。

#### 1. 需求方与开发方的关系

需求方与开发方是对立的统一体，双方均希望将开发项目做好。但需求方可能对信息开发技术缺乏全面的了解；而开发方可能对需求方的需求、细节了解不充分，这使得双方对开发过程的理解存在着差异。这种认识上的差异与理解的不同会导致开发成果与实际需求偏差甚远。因此，项目管理的重要目标便是建立一个便于开发方与需求方之间进行交流的环境。在系统需求分析阶段，开发方与用户方的深入交流是项目获得成功的关键。但这种交流却经常由于双方的各种误解而难以沟通，分析人员总是先把精力集中在整个系统的总需求上，而不会对具体细节作过多的考查，而需求方项目管理人员刚开始往往只能提出粗糙的需求模式。同时因为需求方期望值过高，容易过高地估计计算机软件的能力，认为它一定能实现任何所需功能，所以经常会对所开发的软件大失所望。总之，需求方与开发方的关系是项目管理所要处理的最重要的关系之一，增加沟通和减少误解是处理好这个关系的关键。所以，项目管理人员要有效地安排开发方软件技术人员与需求方系统使用人员进行交流，并保证有畅通的交流渠道。在交流中需求方要尽量避免含糊不清的需求，而开发方要杜绝敷衍了事、得过且过的行为。

#### 2. 需求方项目管理人员与使用人员及决策层的关系

信息系统的使用一方面减轻了工作强度、提高了工作效率，而另一方面也改变了现行的工作管理模式，改变了原有的一些工作流程和工作习惯。但是信息系统的成功与否仍有赖于使用人员的检验。特别是在信息系统的试运行阶段，使用人员对信息系统的使用实际上是对系统的深入测试，有助于帮助开发方进一步完善软件功能，提高软件的实用性、稳定性及可靠性。因此如何鼓励使用人员使用信息系统，帮助他们克服对新的工作模式的畏难情绪也是项目管理的任务之一。任何一种新的工作方式均有其适应及完善过程，需求方的项目管理人员、决策层及使用人员必须充分认识到这一点。当出现问题时，需求方项目管理人员应迅速分析问题，正确判断哪些是由于不适应新的工作模式引起的，哪些是由于操作不当引起的，哪些是由于信息系统本身不完

善引起的，从而对症下药，采取不同的应对措施。

需求方的决策人士是需求方项目管理人员的领导，而行政手段是推行信息系统使用的有力手段之一，他对项目的支持是使信息系统开发成功和顺利实施的保证。因此需求方项目管理人员应随时与决策层沟通，力争取得其鼎力支持。

总之，项目管理人员一方面要时刻注意取得决策层的理解与支持，另一方面要帮助使用人员尽快地适应新的工作方式，及时帮助他们解决使用中遇到的问题，使系统在使用中不断地得以完善。

### 3. 项目管理人员与软件开发人员的关系

项目管理人员与软件开发人员的关系将直接影响软件开发人员的积极性。当使用人员对系统提出问题并要求改动时，软件开发人员往往找出各种理由予以否定，而这正是引起开发方与需求方矛盾的最经常的原因。在信息系统项目开发中，项目管理人员需要经常协调使用人员和软件开发人员的关系，既要满足需求方合理的需求变化，又要充分调动开发人员的积极性。系统分析不够准确、需求方业务需求的改变等诸多因素，均会导致开发方修改系统。作为项目管理人员，应及早提醒开发方注意软件修改的必然性，在系统试运行阶段，将需求方不断提出的需求变动加以归纳整理，集中问题与开发方一起讨论解决方案。在实际运作过程中，需求方管理人员应尽早介入开发工作，及时发现问题，解决问题。这样既满足了需求方对系统改动的需求，又不会因不规则地时常打断开发人员的正常开发工作，使开发人员处于不断的修改状态而失去耐心。

### 4. 系统性能与灵活应变的关系

性能与灵活是系统设计中的一对矛盾，在项目管理中应充分考虑性能与灵活的关系。性能是系统可用性的重要因素，很难想像一个响应速度很慢的系统能得到最终用户的认可。而灵活性是系统适应变化能力的重要因素，一个无法适应工作模式变化的系统也是难以推行的。

## 8.5 信息系统项目质量管理

这一节介绍信息系统质量的概念与特性、信息系统质量控制的组织职能、项目开发的质量控制、信息系统的质量维护。

### 8.5.1 信息系统质量管理概述

信息系统的质量管理是信息系统管理的重要内容。它包括信息系统项目开发的质



量管理和信息系统运行过程的质量管理，还涉及组织机构中各类各层次人员的职责和职能。信息系统的质量管理贯穿于整个信息系统生命周期的全过程，信息系统的管理者们必须对此作全面的考虑。

通常，质量要在一定的范围内定义。一个应用项目的质量与其主要用户、次要用户、操作人员、管理人员和维护人员有关。要达到完美的质量，需花费很高的代价，质量只能是被定义在组织机构的可接受的限度内。质量限度反映出质量不足的影响及达到质量需付出的代价。例如，开发工作的质量不好，可能导致应用项目不能使用，或者还需大量的维修和完善工作。但要一次开发过程就达到系统的完美无缺也是办不到的。

信息系统的质量有许多特性，这与观察的角度有关。从系统的内部来看，系统的质量特性主要是内部结构性能、内部结构的可靠性及结构的连接特性等。从用户的角度来看，系统的质量特性主要有系统的作用、正确性及适应性等。信息系统的质量管理首先是保证信息系统符合用户的质量要求。因此，从用户的外部条件出发可以导出信息系统质量管理的指标，如表 8.4 所示。每个指标的定量值或定性要求则要根据具体的信息系统确定，各个指标的重要程度也有赖于系统的应用领域及其环境。根据各个指标的不同限度，管理者可以对系统作出评审，检验系统是否符合要求，以便进行控制。

表 8.4 信息系统质量管理的指标

质量特性		特性内容	定量性指标举例
可用性	目的性	软件的规格应符合用户要求的特定目的，信息处理（收集、发布等）高速化、提高作业效率、对象地区的范围、适用对象、适用对象项目、系统的运用性等	通过系统化缩短的时间、对象项目数、运用时间、运用周期
	操作性	容易学习软件的使用方法，操作简便	平均学习时间、操作时间、查询次数
	性能	执行特定的功能所需要的计算机资源量和时间	响应时间、屏幕显示时间
正确性	可靠性	应能按规格无故障进行工作	故障件数/工作时间
	准确性	数据的完整性、准确、精确	
	可用性	能够迅速从故障状态恢复正常，减少故障时间，具有故障报告功能、再思考功能、自动恢复功能	恢复时间
	保密性	能够防止没有使用系统内数据资格的人破坏、盗用数据	
适应性	维护性	应能简便地分析软件不良的原因，并进行修正	
	扩充性	应能简便地变更、升级主机或外设	
	兼容性	不改变环境条件即能使用现有功能	
	可移植性	应能把软件简便地移植到其他系统环境，并使之运行	
	连接性	应能简便地与其他系统连接	



### 8.5.2 信息系统质量控制的组织职能

进行信息系统的质量控制，组织保障是最基本的要求，信息系统质量控制的组织职能可以分为3个层次考虑，即组织机构上层管理者的职责、信息系统管理者的职责和系统用户的职责。

#### 1. 组织机构上层管理者的职责

在信息系统的质量控制中，上层管理人员的任务是建立总的组织机构，选择信息系统的负责人，审定计划和预算，并评价其成效。上层管理者的职责主要有如下7个方面：

- (1) 信息系统职能部门的职责和权限范围；
- (2) 选择信息系统的主要管理人员；
- (3) 审定信息系统的计划、长远规划和年度预算；
- (4) 审定主要的硬件和软件系统；
- (5) 审定主要的应用项目；
- (6) 对照计划审查取得的成果并评价信息系统的性能；
- (7) 审查并批准信息系统的质量保证与控制规程。

#### 2. 信息系统管理者的职责

信息系统的主要管理者有组织和监督各种控制和质量保证活动的责任。

(1) 为自己开发的应用项目或者为使用程序软件包之类的应用，建立质量保证规程并监督执行。

(2) 建立并检查信息系统的各种控制职能，如处理控制、存取控制、数据库管理、备份恢复、程序库管理、文件文档管理等。

(3) 项目开发的质量控制。

(4) 建立规程并监督其执行，以便报告关于质量的数据，如误差、故障时间、重新运行、应用项目维护等数据。

#### 3. 系统用户的职责

系统用户是应用系统和数据库开发与维护工作的参与者，他们对质量保证负有责任。特别是当用户自己开发系统时，更应当有具体的质量保证措施。

用户的质量保证责任要求了解数据的关系。了解数据的关系，意味着用户能够识别无效的数据。用户应该负责报告这种可能的无效数据，应设规程来简化这种情况的

报告并对已经更正的结果提供反馈信息。用户还应当对系统的需求定义、输入/输出的结果方式进行确认,这是质量控制的基本责任。

### 8.5.3 项目开发的质量控制

项目开发的质量控制是整个信息系统质量保证的关键,而且系统开发初期的质量管理更为重要。

美国 IBM 公司曾对造成信息系统质量问题的各种错误进行了统计,其结果为:编程错误占 25%,系统分析和设计错误占 45%,程序修改错误占 20%,文档错误占 7%,其他占 3%。

从质量管理的角度来说,错误发现得越早,就越易修改,所花代价就越小。国外曾有人研究后指出,假设在系统分析阶段修正错误所需费用为 1,如果拖到系统设计阶段才修正错误,则至少需要的费用为 4;而到系统运行阶段再修正错误,则所需费用为 30。可见,项目质量控制在一开始就变得十分重要。

项目开发的质量保证包括如下 5 个方面的内容。

(1) 确保获得完整、正确的需求。

(2) 在开发的每一阶段,要休整一下,以进行充分审查并确保该部分工作与系统相协调。

(3) 制定质量控制的程序开发规范,包括系统设计、程序设计、程序检查和程序测试。

(4) 常规的安装调试。

(5) 事后审计评价。

为了保证系统开发的质量,通常需要结合项目的特点,选择恰当的项目开发策略,以对质量加以控制。项目开发策略通常有如下 4 种。

(1) 线性法:也称为常规法,即依照信息系统生命周期各阶段的次序顺序开发,不需经过某些阶段的反复就完成整个系统的开发。这种方法适用于在系统规模较小、问题也不复杂且结构化程度较高的情况下使用。

(2) 线性迭代法:这种方法可以看成是一个反复迭代、求精的过程,即系统开发的各个阶段经过多次反复确认、修改之后,才能保证其符合要求。这种方法是线性法的改进。

(3) 原型法:这是一种试验保证方法,实际上是一种不断的试验、演示、修改,直至完善的项目开发策略。它适用于在用户需求变化多、难以确定的情况下使用。

(4) 复合法:将项目分解成一个个独立的子项目,使各个子项目都能保持相对的独立完整;然后按优先次序进行开发,后面开发的子项目可以利用前面先开发的项目

的成果和经验，每个子项目的开发可以应用前面介绍的几种方法，甚至也可以再次采用复合的方法。这种方法适用于大型、复杂的项目。

项目的规模、项目的结构化程度、用户对信息系统所具备的技能等都对各种策略的选择起着关键作用。

项目的开发策略对项目的质量会产生重要的影响。而项目开发的质量保证还可以通过设置项目质量控制来加以控制。下面介绍各阶段设置质量控制的一些建议。

#### （1）规划阶段

- 决策目标和解决手段是否正确合理？
- 系统结构是否合理？
- 系统资源的可利用性如何？
- 信息系统开发的基础是否确实具备？
- 工程计划安排是否切实可行？

#### （2）系统分析阶段

- 现行系统描述是否正确？
- 新系统功能是否明确？
- 新系统逻辑模型是否合理？
- 子系统的划分是否合理？

#### （3）系统设计阶段

- 模块的划分是否合理？
- 数据结构的设计是否合理？
- 信息规范化程度如何？
- 测试方案和测试用例是否完整？

#### （4）程序设计和测试阶段

- 程序的结构化程度是否高？
- 程序的正确性如何？
- 运行的速度怎样？
- 测试报告是否正确完整？
- 技术指标的考核是否合格？

## 8.6 信息系统开发的文档管理

从信息系统总体规划、系统分析、系统设计到实施应用的整个过程中会形成很多的文档资料，如各种图表、文字说明材料、数据文件、报告等。这些都是未来进行系

统维护、升级或扩展的重要参考资料。可以说,文档管理是信息系统项目管理中非常重要的一部分工作。但是,信息系统的文档管理没有统一的标准或规范。另外,在信息系统建设的实际工作中,有时会因为意识不到文档管理的重要性,而未给予足够的重视,使得工作做得不够细、不够好,为未来的系统维护、扩展等带来了不必要的困难。现在,人们已经开始逐渐认识到系统的文档管理有着非常重要的意义,并且已在实际的信息系统项目管理工作中加强这方面的工作。

### 8.6.1 信息系统的质量维护文档的内容与分类

在信息系统建设过程中涉及的文档类资料很多,资料的格式、内容、载体等都有着很大的区别。为了做好系统的文档管理工作,方便归档和将来使用时的检索,必须对它们进行适当的归类。下面针对技术类文档给出几种文档的分类方法。

按照生命周期法的5个阶段来进行划分,各阶段包含的主要文档如表8.5所示。

在表8.5中,根据生命周期法的5个步骤,给出了信息系统文档的主要内容及分类。这是普遍采用的一种信息系统文档归类方法。

由于信息系统的文档多而杂,所以除了上述归类方法之外,还可以根据格式或载体对信息系统文档进行划分。按照这种划分方法,信息系统文档又可分为原始单据或报表、磁盘文件、文件打印件、大型图表、重要文件原件、光盘存档等几大类。

#### 1. 原始单据或报表

在信息系统的调查分析阶段会获取大量的原始单据和原始报表。这类资料一般都是以纸张为存储介质的,其大小、格式一般都没有统一的标准,容易散落、破损及丢失,如入库单、领料单、过秤单、原材料台账、生产日报等。对这类文档资料应编好目录,装订成册。如果需要,则可以同时复印并装订一个副本。

#### 2. 磁盘文件

磁盘文件是目前信息系统文档最主要的存储方式。由于计算机办公软件的普遍使用,所以各类报告或说明书一般都是通过使用文字处理、幻灯片制作等软件工具生成的(例如,采用软件工具WPS、Word、Excel、PowerPoint等),如可行性研究报告、系统分析说明书、系统设计说明书、程序设计说明书等一般都采用这种方式编写和保存。磁介质的文档资料虽然占用空间小、信息量大、易于保管,但如果磁盘发生损坏,则会引起数据的彻底丢失。因此,需要做好备份工作,例如,可以刻成光盘备份等。

表 8.5 信息系统开发各阶段的文档

阶 段	文 档	相关内容
1. 系统规划	① 可行性研究报告 ② 系统开发计划	01. 项目背景 02. 系统目标及总体功能需求和关键信息需求 03. 系统可行性分析 04. 开发进度
2. 系统分析	① 系统分析报告	01. 组织结构及人员配备 02. 组织职能划分及与其他部门的关系 03. 业务及相关数据调查表 04. 业务及相关数据原始单据和报表 05. 调查记录和整理结果 06. 业务流程图，用例图 07. 数据流程图 08. 数据字典 09. U/C 矩阵图 10. 管理模型及相应的计算关系公式 11. 各种图表的辅助文字说明 12. 目标系统的逻辑功能结构
3. 系统设计	① 总体设计报告 ② 详细设计报告	01. 目标系统的硬件配置方案 02. 目标系统的系统软件配置方案 03. 目标系统的业务流程描述 04. 目标系统的数据类描述 05. 目标系统的功能结构 06. 数据库文件的设计 07. 安全保密机制 08. 编码方案 09. 功能模块的输入/输出设计 10. 功能模块的处理流程
4. 系统实施	① 程序设计说明书 ② 源程序备份文件 ③ 系统测试报告 ④ 用户使用手册	01. 变量说明 02. 程序处理流程 03. 程序间的调用关系 04. 使用的数据库文件 05. 公共程序等的特殊功能说明 06. 测试环境、数据准备 07. 测试时间、人员安排 08. 测试结果 09. 用户培训计划 10. 系统使用说明 11. 系统试运行阶段的试运行和修改记录
5. 系统运行 维护与评价	① 系统运行日志 ② 系统修改与维护报告	01. 系统运行阶段的运行记录 02. 系统运行阶段的维护和修改记录 03. 系统的评价或鉴定结果

### 3. 文件打印件

文件打印件与电子文件是同时存在的，这主要是出于交流和使用上的方便。对于这些打印出的文档，应该装订成册，切忌散页存放，以免部分丢失。另外，各种报告和说明书都有一个反复修改的过程，要注意区分修改前的版本和修改后的版本，避免混淆带来的使用上的不便，甚至出现错误。

### 4. 大型图表

在信息系统文档中，还可能出现一些大型图表。例如，在大型企业的信息系统建设过程中用到的 U/C 矩阵图、E-R 图等。有时这类图表的大小可以占用一面墙。由于这些图表需要折叠存放，因此在绘制时，一定要选择不易被折断的纸张，在保存时，需要放在档案袋或档案盒里，以免磨损。

### 5. 重要文件原件

信息系统的文档主要是技术文档，但也有一些涉及权利义务关系的重要文件，如项目合同或协议书、系统验收或评审报告等。

### 6. 光盘存档

光盘存档是近几年发展起来的文档保存方式。由于光盘的存储量大、体积小，所以光盘存档受到了普遍的欢迎。

## 8.6.2 文档的规范化管理

在 8.6.1 节中，介绍了信息系统文档的内容和分类。下面讨论文档规范化管理的方法。

目前，信息系统文档管理方面尚没有统一的标准，在具体工作中也没有固定的模式。但是，在一个信息系统项目的整个运作过程中，必须有一个统一的内部标准，并应该严格执行。信息系统文档的规范化管理主要体现在文档书写规范、图表编号规则、文档目录编写标准和文档管理制度等几个方面。

#### 1. 文档书写规范

信息系统的文档资料涉及文本、图形、表格等多种类型，无论哪种类型的文档都应该遵循统一的书写规范，其中包括符号的使用、图标的含义、程序中注释行的使用、注明文档书写人及书写日期等。例如，在程序的开始，要用统一的格式包含程序名称、程序功能、调用和被调用的程序、程序设计人等。

2. 图表编号规则

在信息系统的开发过程中会用到很多的图表，对这些图表进行有规则的编号，可以方便图表的查找。图表的编号一般采用分类结构。根据生命周期法的5个阶段，可以给出如图8.3所示的图表分类编号规则。根据该规则，就可以通过图表编号判断：该图表处于系统开发周期的哪一个阶段，属于哪一个文档、文档中的哪一部分内容及第几张图表等。对照8.6.1节中对系统文档的分类就可以知道，图表编号2—1—08—02对应的是系统分析阶段系统分析报告中数据字典的第2张表。

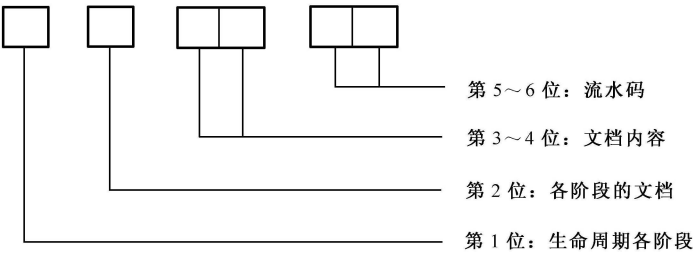


图 8.3 图表分类编号规则

3. 文档目录编写标准

为了存档及未来使用的方便，应该编写文档目录。信息系统的文档目录中应包含文档编号、文档名称、格式或载体、份数、每份页数或件数、存储地点、存档时间、保管人等。文档编号一般为分类结构、可以采用与图表编号类似的编号规则。文档名称要书写完整规范。文档格式或载体指的是原始单据或报表、磁盘文件、磁盘文件打印件、大型图表、重要文件原件、光盘存档等。信息系统文档目录的编写可以采用表8.6所示的形式。

表 8.6 ×××信息系统文档目录

编号	文档名称	文档格式或载体	份数	页数	存储地点	存档日期	保管人
1—1	可行性研究报告	软盘	2	1	507 档案柜	2007/2/9	谷月
1—2	系统开发进度	软盘	2	1	507 档案柜	2007/2/9	谷月
2—1	系统分析说明书	软盘	2	1	507 档案柜	2007/2/9	谷月
2—1—04	业务原始单据和报表	原始单据或报表	1	56	507 档案柜	2007/2/9	徐圆梦
2—1—09	U/C 矩阵图	大型图表	1	1	507 档案柜	2007/2/9	徐圆梦
...	...	...	...	...	...	...	...
5—2—03	系统鉴定报告	重要文件原件	1	3	506 档案柜	2007/3/9	汪涛

4. 文档管理制度

为了更好地进行信息系统文档的管理，应该建立相应的文档管理制度。文档的管理制度需根据组织实体的具体情况而定，一般包括建立文档的相关规范、文档借阅记录的登记制度、文档使用权限控制规则等。建立文档的相关规范是指文档书写规范、图表编号规则和文档目录编写标准等。文档的借阅应该进行详细的记录，并且需要考虑借阅人是否有使用权限。当文档中存在商业秘密或技术秘密的情况时，还应注意保密问题。

习 题

- 1. 项目管理知识体系的 9 个知识领域是哪些？
- 2. 信息系统的项目角色是什么？
- 3. 简述信息系统项目时间管理的流程。
- 4. 项目管理的五项基本内容是什么？
- 5. 信息系统建设项目的特点是什么？
- 6. 信息系统项目实施过程中的项目管理步骤有哪些？
- 7. 试说明文档在信息系统质量管理中的作用和意义。
- 8. 设某信息系统项目的任务分解及工期和依赖关系如下表所示，试绘出该项目从今天开始启动的活动网络图（可以不考虑休息日）。

任务	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>	T <sub>9</sub>	T <sub>10</sub>	T <sub>11</sub>	T <sub>12</sub>	T <sub>13</sub>
工期（天）	7	11	12	21	14	9	12	11	7	9	11	2	8
依赖		T <sub>1</sub>			T <sub>2</sub>	T <sub>2</sub> T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub> T <sub>6</sub> T <sub>4</sub>	T <sub>3</sub> T <sub>7</sub>	T <sub>5</sub>	T <sub>10</sub> T <sub>9</sub>	T <sub>8</sub>	T <sub>7</sub>





# 参 考 文 献

- [1] Surajit Chaudhuri, Umeshwar Dayal, An overview of Data Warehousing and OLAP Technology, SIGMOD Record, Vol. 26, No. 1, March 1997
- [2] George Colliat, OLAP, Relational, and Multidimensional Databases Systems, SIGMOD Record, Vol. 25, No. 3, September 1996
- [3] Edward Yourdon, Cal Argila, Case Studies in Object-Oriented Analysis & Design, Prentice Hall Inc. 1996
- [4] Ronald J. Norman, Object-Oriented Analysis and Design, Prentice Hall Inc. 1996
- [5] Specialized Requirements for Relational Data Warehouse Servers, A Red Brick Systems White Paper, September 1995
- [6] Star Schemas and STARjoin Technology, A Red Brick Systems White Paper, September 1995
- [7] E. F. Codd, S. B. Codd, and C. T. Salley, Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate, E. F. Codd & Associates, 1993
- [8] Zachman, J. "A Framework for information Systems Architecture," IBM Systems Journal, Vol. 26, No. 3, White Plains, New York, 1986
- [9] Coad P. Yourdon E. Object-Oriented Analysis, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1991
- [10] Coad P. Yourdon E. Object-Oriented design, Englewood Cliffs, NJ: Prentice Hall, 1991
- [11] 张维明, 宁枫. 计算机信息系统分析与设计. 长沙: 国防科技大学出版社, 1990
- [12] 张维明, 肖卫东, 杨强等. 信息系统工程. 北京: 电子工业出版社, 2003
- [13] 张维明, 邓苏, 罗雪山. 信息系统建模技术与应用. 北京: 电子工业出版社, 2002
- [14] 邓苏, 张维明, 汤大权等. 信息系统集成技术. 北京: 电子工业出版社, 2002
- [15] 韩万江, 姜立新. 软件开发项目管理. 北京: 机械工业出版社, 2004
- [16] Craig Larman. 李洋等译. UML 和模式应用: 面向对象的分析与设计导论. 北京: 机械工业出版社, 2006
- [17] Grady Booch. 冯博琴译. 面向对象分析与设计. 北京: 机械工业出版社, 2003
- [18] 邵维忠, 杨芙清. 面向对象的系统设计. 北京: 清华大学出版社, 2003
- [19] 薛华成. 管理信息系统. 北京: 清华大学出版社, 2003
- [20] 苏选良. 管理信息系统. 北京: 电子工业出版社, 2003
- [21] 陈文伟, 黄金才, 赵新昱. 数据挖掘技术. 北京: 北京工业大学出版社, 2002
- [22] 罗超理, 李万红. 管理信息系统原理与应用. 北京: 清华大学出版社, 2002
- [23] Sami Zahran. 陈新, 罗劲枫译. 软件过程改进. 北京: 机械工业出版社, 2002
- [24] 史忠植. 知识发现. 北京: 清华大学出版社, 2002

- [25] 左美云, 邝孔武. 信息系统的开发与管理教程. 北京: 清华大学出版社, 2001
- [26] J. W. Han, M. Kamber. 数据挖掘概念与技术. 北京: 机械工业出版社, 2001
- [27] Ralph M. Stair. 信息系统原理. 北京: 机械工业出版社, 2000
- [28] 左美云, 邝孔武. 信息系统开发与管理教程. 北京: 清华大学出版社, 2006
- [29] 赵瑞莲. 软件测试. 北京: 高等教育出版社, 2004
- [30] 张友生等. 全国计算机技术与软件专业技术资格(水平)考试辅导教程 系统架构设计师教程. 北京: 电子工业出版社, 2006
- [31] 俞星. 基于 J2EE 和 .NET 平台的 Web 应用开发的比较与研究. 浙江大学硕士学位论文, 2007
- [32] 王黎, 袁永康译. Microsoft .NET 战略. 北京: 清华大学出版社, 2002
- [33] 邵维忠, 杨芙清. 面向对象的系统分析. 北京: 清华大学出版社, 1998
- [34] 周广声, 李新月, 杨丽萍. 信息系统工程原理、方法及应用. 北京: 清华大学出版社, 1998
- [35] 汤庸. 结构化与面向对象软件方法. 北京: 科学出版社, 1998
- [36] 王珊. 数据仓库技术与联机分析处理. 北京: 科学出版社, 1998
- [37] 李之棠, 李汉菊. 信息系统工程原理、方法与实践. 武汉: 华中理工大学出版社, 1997
- [38] 齐治昌, 谭庆平, 宁洪. 软件工程. 北京: 高等教育出版社, 1997
- [39] 贾鑫, 卢昱. 模糊信息处理. 长沙: 国防科技大学出版社, 1996
- [40] 黄梯云. 管理信息系统. 北京: 电子工业出版社, 1995
- [41] 吴泉源, 刘江宁. 人工智能与专家系统. 长沙: 国防科技大学出版社, 1995
- [42] 王众托等. 计算机决策支持系统. 北京: 中国石化出版社, 1995
- [43] 王克宏, 汤志忠. 知识工程与知识处理系统. 北京: 清华大学出版社, 1994
- [44] 薛华成. 管理信息系统. 北京: 清华大学出版社, 1993
- [45] 姜旭平. 信息系统分析——概念、结构、机理、分支与发展. 长沙: 湖南科学技术出版社, 1993
- [46] 董大钧. SAS 统计分析软件应用指南. 北京: 电子工业出版社, 1993
- [47] 王珊, 罗力. 从数据库到数据仓库. 中国人民大学数据与知识工程研究所. 计算机世界报, 1996
- [48] 邹生等. 信息系统工程概论. 北京: 中国计划出版社, 1993
- [49] 蔡希尧, 陈平. 面向对象技术. 西安: 西安电子科技大学出版社, 1993
- [50] 张海藩. 软件工程导论. 北京: 清华大学出版社, 1992
- [51] 冯师道. 管理信息系统. 北京: 科学出版社, 1992
- [52] 诸葛海, 洪树春等. 信息系统与类比方法论. 杭州: 浙江大学出版社, 1992
- [53] 汪成为等. 面向对象的分析、设计及应用. 北京: 国防工业出版社, 1992
- [54] 夏安邦. 决策支持系统引论. 上海: 同济大学出版社, 1991
- [55] 张兆基, 易进先, 李希平. 系统工程. 北京: 知识出版社, 1991
- [56] 颜永琪等译. 决策支持系统基础. 福州: 福建科学技术出版社, 1989
- [57] 陈圣国. 信息系统分析与设计. 西安: 西安电子科技大学出版社, 2001